WL-TR-95-1105

STANDARD ANALYZER OF VHDL APPLICATIONS FOR NEXT GENERATION TECHNOLOGY (SAVANT)

PRAVEEN CHAWLA
PHILIP A. WILSEY
HERBERT L. HIRSCH
JEFFREY CARTER

MTL SYSTEMS, INC.
3481 DAYTON-XENIA ROAD
DAYTON OH 45432-2796

APRIL 1995

FINAL REPORT FOR 06/23/94-04/24/95

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Al Scarpelli

AL SCARPELLI, Project Engineer
Data & Signal Processing Section
Information Processing
Technology Branch

John W. Hines

JOHN W. HINES, Chief
Design Branch
Microelectronics Division

Stanley E Wagner

STANLEY E. WAGNER, Chief
Microelectronics Division
Solid State Electronics
Directorate

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/AAAT-2 , WPAFB, OH 45433-7409 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 1995	FINAL 06/23/94--04/24/95

4. TITLE AND SUBTITLE
STANDARD ANALYZER OF VHDL APPLICATIONS FOR NEXT NEXT GENERATION TECHNOLOGY (SAVANT)

5. FUNDING NUMBERS
C F33615-94-C-1469
PE 65502
PR 3005
TA 06
WU 57

6. AUTHOR(S)
PRAVEEN CHAWLA HERBERT L. HIRSCH
PHILIP A. WILSEY JEFFREY CARTER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MTL SYSTEMS, INC.
3481 DAYTON-XENIA ROAD
DAYTON OH 45432-2796

8. PERFORMING ORGANIZATION REPORT NUMBER
MFR-93-006/CSF325

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-95-1105

11. SUPPLEMENTARY NOTES
THIS IS A SMALL BUSINESS INNOVATION RESEARCH REPORT (SBIR), PHASE 1

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

THIS WORK ESTABLISHED THE FEASIBILITY OF A STANDARD MEDIUM (INTERMEDIATE FORM OR IF) FOR THE EXCHANGE OF VHDL-ENCODED DATA AMONG COMPUTER-AIDED DESIGN (CAD) SYSTEMS. THE CURRENT PROLIFERATION OF VENDOR-PROPRIETARY, NON-STANDARD IFs AND ANALYZERS FORCES USERS TO EITHER USE ALL THE CAD TOOLS FROM ONE VENDOR OR PURCHASE SEVERAL VHDL ANALYZERS, ONE FROM EACH DISTINCT CAD TOOL VENDOR. THE SOLUTION TECHNOLOGY WAS NAMED THE STANDARD ANALYZER OF VHDL APPLICATIONS FOR NEXT-GENERATION TECHNOLOGY (SAVANT). THE WORK FOCUSED ON THREE OBJECTIVES, TO: 1)ESTABLISH THE TECHNICAL FEASIBILITY OF THE SAVANT TECHNOLOGY, 2)ESTABLISH COMMUNITY ACCEPTANCE OF THE TECHNOLOGY, AND 3)PRODUCE PRELIMINARY DESIGN CONCEPTS FOR PHASE II IMPLEMENTATION. THE TECHNOLOGY WAS FOUND TO BE A VALID, FEASIBLE, COMMERCIALIZABLE SOLUTION. THE EFFORT DEMONSTRATED THE IF CONCEPTS, A PROTOTYPE STANDARD ANALYSER, COMPILER SUPPORT TOOLS, AND RECORD AND PLAYBACK FORMATS. THE WORK ALSO DEFINED THE SAVANT DOCUMENTATION FORM, IDENTIFIED A DISTRIBUTION SCHEME, BEGAN TO STIMULATE DESIGN COMMUNITY ACCEPTANCE, AND PRODUCED A COMMERICAL PRODUCT INSERTION PLAN. POTENTIAL APPLICATIONS INCLUDE MYRIAD RESEARCHERS, DEVELOPERS, AND DESIGN TOOL PRODUCERS INVOLVED WITH CAD IN VHDL.

14. SUBJECT TERMS			15. NUMBER OF PAGES
VHDL	VHDL SIMULATION	INTERMEDIATE FORM (IF)	70
CAD	VHDL ANALYZER	EDA	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

SAVANT

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

PREFACE

This report describes a Phase I Small Business Innovation Research (SBIR) project entitled "Solid State Electronics Directorate Applied Research." The program was conducted for the United States Air Force Wright Laboratory Solid State Electronics Directorate (USAF-WL/ELED), from June, 1994 through April, 1995, under contract number F33615-94-C-1469. The research was conducted by MTL Systems, Inc., Dayton, Ohio, and the University of Cincinnati (UC) Department of Electrical and Computer Engineering and Science, Cincinnati, Ohio. The MTL project number and title were CSF325, "Standard Analyzer of VHDL Applications for Next-Generation Technology," or "SAVANT."

The MTL Principal Investigator was Dr. Praveen Chawla, with additional MTL support provided by Mr. Jeff Carter and Mr. Herb Hirsch. The UC effort was conducted by Dr. Philip Wilsey. Mr. Al Scarpelli and Captain Scott Bilik were the USAF Project Engineers. The authors wish to express their thanks to Mr. Scarpelli and Captain Bilik for the attentive support and guidance provided to the MTL project team, which contributed significantly to the success of this effort.

1.0 INTRODUCTION

This report documents the results of a Phase I Small Business Innovation Research (SBIR) project entitled "Standard Analyzer of VHDL Applications for Next-Generation Technology" (SAVANT). The effort was conducted under United States Air Force SBIR Topic Number AF94-134, "Solid State Electronics Directorate Applied Research."

We have organized this report to provide ready access to pertinent information for a variety of readers' needs. In the remainder of this introductory section (Section 1), we provide a succinct overview of the project, in the context of problem, background, Phase I objectives and requirements, and Phase I results. Readers who require a concise "snapshot" of the project will find this section especially useful. Next, in Section 2, we provide an elaborated discussion of our technical investigations, with attention to the particular technical achievements and their relevance to meeting the project requirements and objectives. Readers with interest in the technical issues will find these details in this section. Then, in Section 3, we present our Phase I results, in the context of their extent toward solving the problem and forming a foundation for subsequent Phase II activity. Here, readers may ascertain exactly how far our Phase I work has brought us toward a solution to the fundamental problem. Finally, in Section 4, we offer our conclusions and recommendations for a subsequent Phase II program. From this information, readers may evaluate how the multi-phase SBIR program can provide (1) the particular technology innovation to solve the technical problem at hand, and (2) a viable commercial product, thus meeting the goals of the SBIR program in general.

1.1 The Problem and Significance

Here, we describe the problem which we attacked in Phase I, as well as its significance. In this context, "significance" relates to the benefits to be realized by the electronic design automation community at large from solving the problem, as we shall explain.

The *principal problem* addressed by this effort is the absence of an established, standard Intermediate Form (IF) for the exchange of VHDL-encoded electronic data among Computer-Aided Design (CAD) systems. The result of this

absence is apparent within the presently-constrained basic research environment and sub-optimal nature of CAD-in-VHDL tool development. Consider how VHDL is applied in such tool development. VHDL presents a standard format for human comprehension or encoding of digital system designs. Analyzing and processing VHDL source code is quite difficult and requires considerable effort. Furthermore, there is a broad range of complex CAD tools that are generally available to support the computer system design process, and each distinct CAD tool must input design data encoded in VHDL.

Presently, most CAD tool vendors must execute a cumbersome process to realize a CAD-in-VHDL product. Typically, they (1) design an in-house intermediate form (IF); (2) build a VHDL analyzer that validates the (static) correctness of the input VHDL, producing an IF representation of the input; and (3) input the IF to each in-house-developed CAD tool. In other words, processing VHDL as the source input language places additional, unnecessary burden upon the construction of such CAD tools. The result is a tightly-coupled, non-standard analyzer and IF, within a particular, vendor (application)-specific environment, as shown in Figure 1.

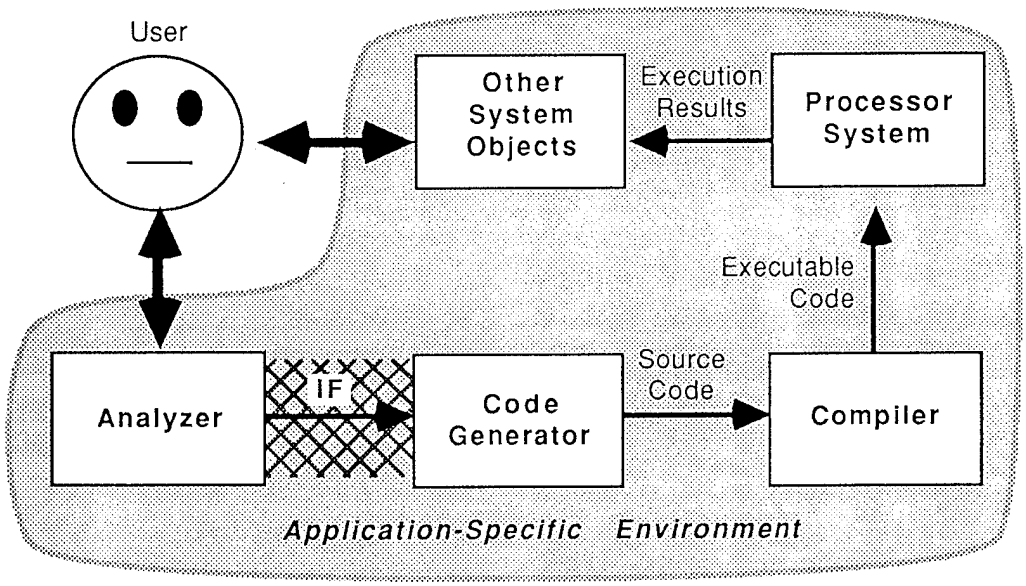

Figure 1. The Problem - Tightly-Coupled, Vendor-Specific CAD-in-VHDL

A resulting problem is the current proliferation of several of these vendor-proprietary, non-standard IFs and analyzers. In rare instances, a vendor may sell (at high cost) the IF and analyzer to a third party. Unfortunately, no vendors are currently willing to standardize (and fully productize) their IF. Consequently, each vendor maintains an internal IF and markets both tools that use the IF and

VHDL analyzers to produce the IF. Consequently, users are forced to either use all the CAD tools from one vendor or purchase several VHDL analyzers, one from each distinct CAD tool vendor whose design tools are being used.

Another aspect of the problem is that this lack of a widely-available, standard IF for VHDL also inhibits basic research. That is, before embarking upon a research investigation in CAD with VHDL, researchers must either design their own particular IF and build an analyzer to translate VHDL to the IF, or they must purchase a vendor-supplied VHDL analyzer/intermediate form (A/IF). The former approach is expensive in time and effort, and generally results in an inferior VHDL analyzer/CAD system that operates only over a limited VHDL subset. The latter approach is subject to the nature of the chosen A/IF. It consequently suffers from high cost and the research project is subject to any changes in the IF produced by the vendor to support their internal tool development. Furthermore, because the IF is generally not a primary product for the vendor, documentation and support tools are generally of poor quality.

The purpose of SAVANT is to directly mitigate these problems. Its *significance* will be that of a community-wide improvement in tool compatibility, as well as a significant enhancement to the overall effectiveness of basic CAD-in-VHDL research and development. We consider SAVANT to consist of two principal components, the IF and analyzer, and a supporting component, the record/playback tool for archiving the IF in file form, as illustrated in Figure 2. The standard IF will provide a common internal representation that vendors can follow and to which users can request adherence. Furthermore, the availability of a public domain analyzer and library subsystem will dramatically promote additional research and development in CAD and its integration with VHDL. Finally, source code availability will also enable and promote integration and cross-coupling between other design language efforts. For example, there is currently an effort underway, sponsored by USAF Rome Laboratory, to develop a standard *analog* hardware description language (VHDL-A). This VHDL-A effort could extend SCRAM (SAVANT's VHDL analyzer) to support the additional features of analog description and promote a rapid integration of VHDL-A technology with VHDL technology among vendors and users alike.

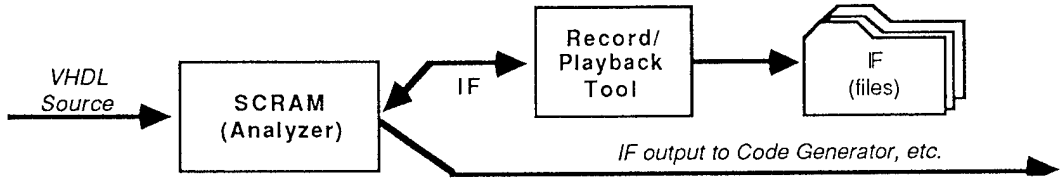

Figure 2. Principal components of SAVANT

Having discussed the problem and its significance, we may now turn to a description of the background from which it evolved, which is the subject of the next section.

1.2 The Background

In our background discussions of this section, we describe the evolution of our problem to the point at which we began our Phase I effort. As this particular problem involves both technical and business issues in the Electronic Design Automation (EDA) industry, our discussions address both these aspects.

From a technical point of view, a public-domain analyzer and a standard IF have always made good sense. Obviously, such standardization promotes a more open-system environment, encourages and facilitates research and development, and generally works to the benefit of users and tool developers alike. However, due to the lack of standardization as VHDL was beginning its surge toward pre-eminence as a digital HDL, tool vendors were forced to make early design choices, to provide products for the growing market, without the benefit of such standardization.

The result of these design choices was an early divergence of analyzers and IFs among the several VHDL analyzer vendors. However, as the market demands were high, and the analyzers were being accepted and selling well into the user community, there was no particular incentive for analyzer developers to adhere to some standard. This momentum has carried the industry to the state in which we find it today.

However, the environment is now changing. First of all, CAD-in-VHDL has proliferated to the point where individual organizations are using a variety of CAD-in VHDL tools, and are directly experiencing the problems associated with disparate, vendor-specific analyzers. Second, there is more competition among tool vendors, and the toolmaker whose products will work across the widest variety of VHDL analyzers will most likely enjoy the most success. Finally, there is the demand for more VHDL research, and an abundance of researchers willing to do it, except they cannot afford several, different analyzers. Obviously, something has to yield. Either (1) the user and tool developer community has to concede that tools and researchers must resolve themselves either to limited environments or the burden of multiple, specialized interfaces to the several analyzers, or (2) the analyzer community has to accept standardization and seek their rewards in support environments.

Based upon the problem and background we have described, we established

certain goals and objectives for our Phase I effort, to properly focus our effort. These are the subjects of the next section.

1.3 The Phase I Objectives, Requirements, and Task Plan

Our Phase I effort was specifically focused upon establishing the feasibility of a solution to the problem, forming a solid foundation for both Phase II work, and realizing a viable commercial product. We now describe the Phase I goals and objectives we established to achieve this focus, and the task plan we implemented to accomplish these ends.

The Objectives: The stated, general objective of the original solicitation was to "Explore innovative technologies and demonstrate feasibility." However, we required more specific objectives to solve our specific problem. In forming our Phase I objectives, we considered the problem at hand, as articulated in Section 1, as well as several other issues. The first issue was feasibility. Hence, one of our objectives had to answer the question "Is SAVANT feasible?" Here, we considered feasibility in the context of both technology and community acceptance. SAVANT needed to be both producible in today's technology and viewed as a necessary tool by users to be considered feasible. The second issue was commercialization potential. Our research had to determine whether or not a viable, marketable product could be derived from SAVANT, given that it was indeed feasible. This aspect also included technology and user acceptance aspects, since products derived from SAVANT needed to be producible at a competitive cost and desired by users to be considered marketable. A final issue was that of scope. We had to set objectives which could reasonably be achieved within the resources of a Phase I effort, and in doing so we needed to consider how these objectives would support a smooth transition into a Phase II program. In consideration of all these aspects, we established our Phase I objectives as follows:

Objective 1 - Establish the technical feasibility of the SAVANT technology as a standard A/IF exchange medium for CAD in VHDL. In meeting this objective, we needed to produce quantified technical investigation results which would confirm that the innovations represented by SAVANT could be constructed in today's technology. Its achievement would specifically address the chief problem (lack of a standard A/IF) stated in Section 1.

Objective 2 - Establish community acceptance of the SAVANT technology and Define the Commercial Product. Here, we wished to obtain valid community endorsement of, and desire for, the SAVANT technology, should it indeed be implemented. We also wished to provide definition of what portion of the

SAVANT technology may be effectively transitioned into a commercial product. In achieving this objective, we expected to address the ancillary problem of constrained (by lack of a standard A/IF) basic research for CAD-in-VHDL, discussed in Section 1.

Objective 3 - Produce valid preliminary design concepts for the two principal elements of SAVANT: the IF and the Analyzer. By achieving this objective we planned to produce a solid foundation for Phase II development. Its achievement would further support the feasibility and commercialization aspects, by showing the beginning of a clear path to development and subsequently to productization and proliferation of SAVANT within the community.

In achieving these objectives, as we document in this report, we established a problem-responsive, feasible, commercializable basis for SAVANT, and produced the preliminary design elements from which a Phase II program may be initiated.

The Project Performance Requirements: Certain performance requirements for the SAVANT program were designed, to bridge the problem and performance domains. These specific Project Performance Requirements (PRs), were defined to particularly ensure that the program objectives were achieved, and to provide proper definition for tasking within our program plan. These PRs and the objectives they were designed to support were:

PR	SUPPORTS ACTIVITY	TO ACHIEVE OBJECTIVE
1.	Establish a preliminary standard intermediate form (IF)	1,3
2.	Establish willingness of CAD community to accept proposed standard	2
3.	Establish vendor community participation	2
4.	Reactivate DASC subcommittee	2
5.	Select compiler support tools to enable construction of SCRAM	1,3
6.	Demonstrate the capability of the compiler support tools by producing a "prototype" SCRAM	1,3
7.	Establish initial file format for record/playback	1,3

Performance Requirements 1, 5, 6, and 7 were designed to achieve Objective 1, to establish the technical feasibility of the SAVANT technology as a standard A/IF exchange medium for CAD in VHDL. By establishing the preliminary form, selecting tools and experimentally confirming the ability to compile and construct the analyzer (SCRAM), and establishing the record/playback file

format, we planned to confirm the format, constructability, and exchange medium capability of the SAVANT technology.

Performance Requirements 2, 3, and 4 were designed to achieve Objective 2, to establish community acceptance of the SAVANT technology. By gaining the acceptance of the CAD community and participation of the vendor community, we expected to define the basis for full community acceptance and definition of the commercializable aspects within SAVANT. Furthermore, by reactivating a subcommittee within the Design Automation Standards Committee (DASC), we expected to establish the formal basis for ensuring continued community participation as we develop SAVANT through Phase II and undertake Phase III or other commercialization activities.

Performance requirements 1, 5, 6, and 7 were also designed to achieve Objective 3, to produce valid preliminary design concepts for the two principal elements of SAVANT: the IF and the analyzer. These design concepts, encompassing the preliminary standard IF, compiler tools, and record/playback file format would establish the necessary basis of a preliminary design. They would validate (1) the nature of IF itself, (2) the ability to construct the analyzer, and (3) the file exchange/archiving format. In other words, this foundation for further development and commercialization was expected to prove that the format is proper and achievable, that the compiler is constructable, and that the exchange format is proper and established.

The Task Plan: Summarily, the performance requirements just described were designed to specify the proper tasking within the program to ensure that the program objectives are met. The resulting task plan needed to provide for cohesive interaction among tasks necessary to produce products which will satisfy the program performance requirements. Our plan to achieve this interaction consisted of 6 tasks, integrated as illustrated in Figure 3.

In Task 1, we planned to apply our proposed concepts for the standard IF, which we presented in our Phase I proposal. The result of this task was to be first a "draft" standard IF definition, which would be shared with the design community under Task 2, and iterated into the Phase I-level standard IF definition which is a deliverable product of our Phase I effort. This definition, which would satisfy Performance Requirement 1 (Establish preliminary standard IF) was to be completed to a preliminary design-level of detail.

Under Task 2, we planned to conduct interactions with the community, including users, vendors, and particularly the DASC. The result of this task, although not an explicit project deliverable, would be the useful knowledge and

community acceptance necessary to ensure a valid and acceptable standard IF. Additionally, this task would obtain community needs for a SAVANT Support Environment, which we viewed as the commercial product to be derived from this program. Task 2 would satisfy Performance Requirements 2, 3, and 4 (Establish community acceptance, vendor participation, DASC subcommittee). However, in the course of the program, this task's focus was modified. In concert with our WL/ELED sponsor, we decided to limit our interaction with the community to VHDL International Users' Forum (VIUF) meetings, postponing more active interaction to Phase II.

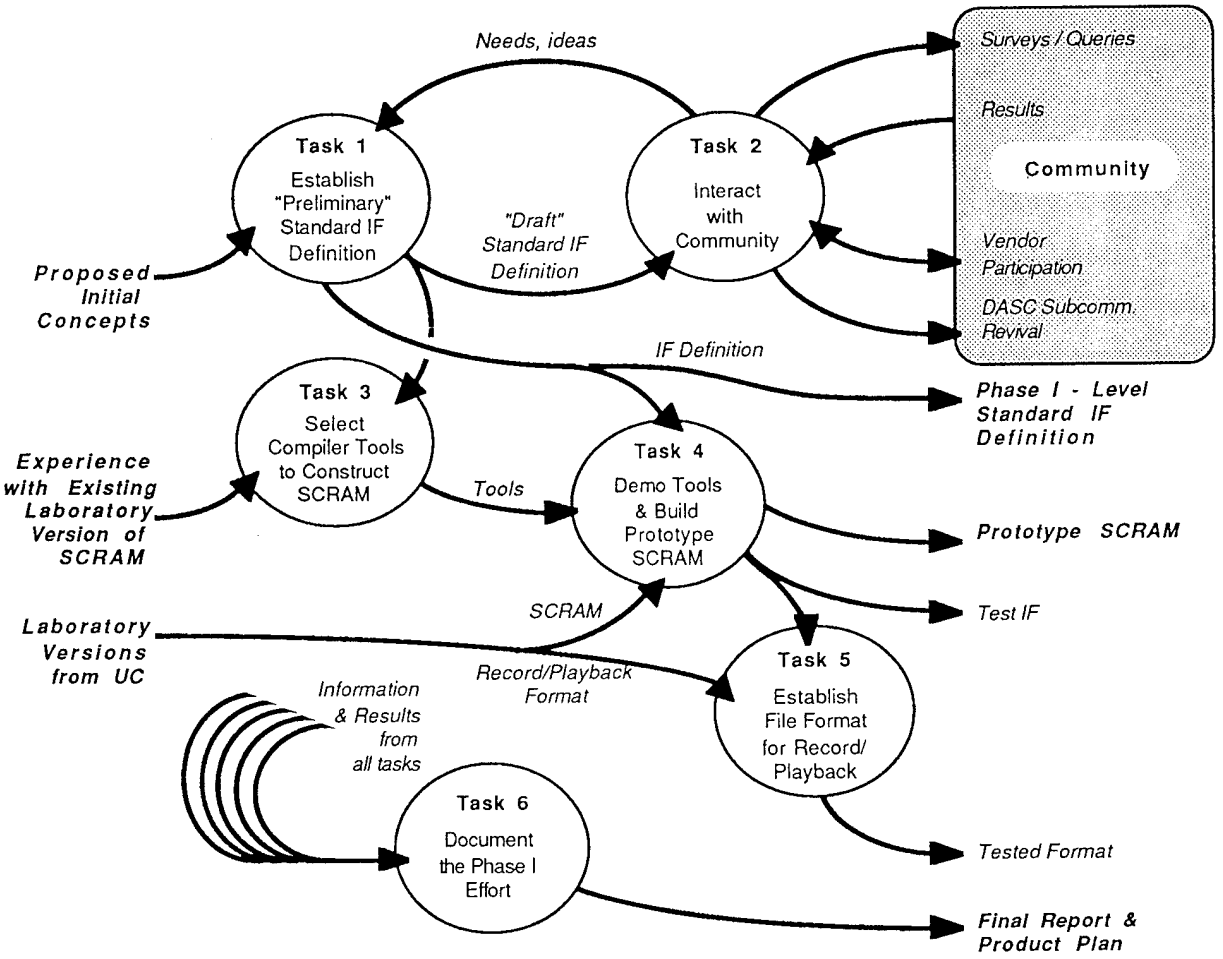

Figure 3. Phase I Task Plan

Tasks 3 and 4 were to be a tightly-integrated pair. Together, their execution would satisfy Performance Requirements 5 and 6 (Select tools and construct prototype SCRAM). In Task 3, we would begin with our experience with the laboratory version of SCRAM at UC, and the draft standard IF definition from

Task 1. Under this task, we would select the appropriate compiler tools to construct the deliverable prototype SCRAM under Phase I. Then, in Task 4, we would proceed to develop and test the prototype SCRAM, supplementing it with the final standard IF definition produced by Task 1, as the task proceeds. The results of Task 4 would be the deliverable prototype SCRAM and the test IF used to confirm its functionality, documented to a preliminary design-level of detail.

In Task 5 we were to establish the record/playback format, to satisfy Performance Requirement 7. The result was to be a tested format capable of supporting the archiving needs of the IF, and documented to a preliminary-design level of detail.

The documentation aspect of the Phase I effort evolved into a two-faceted endeavor. Under Task 6, we originally planned to assimilate all information regarding designs, discoveries, and community interaction into this document, a Phase I Final Report. This report was to also contain a product insertion plan for transfer of the SAVANT technology into the commercial sector. Additionally, while executing the other Phase I tasks, we became aware of the need to define some tools, formats, translation means, and an overall framework for the ultimate production and distribution of supporting documentation for the Phase II SAVANT implementation. We felt that since such documentation support definition would be a critical factor in a successful Phase II endeavor, we should expend some Phase I resources to achieve it. Hence, we added a task (un-numbered) to define the documentation framework and elements, and proceeded to evaluate documentation support candidates.

In summary, our requirements and objectives were well-focused upon the problem, in careful consideration of its background and our available resources, and the task plan was designed to support them. In actuality, as is usually the case with research, the task activities and results from the project, although successful, diverged somewhat from what we originally expected. The added documentation definition task described above is a particular example of this, and there were other instances as well. In Section 1.4, which follows, we briefly summarize the results. Then, in Sections 2 and 3 we elaborate the particulars regarding our tasks, activities, and results, duly noting divergences from the original plan.

1.4 The Phase I Summary Results

In this section, we provide a summary statement of our Phase I results. Although we offer an elaborated description of these results in Section 3, we wanted to complete our Section 1 project "snapshot" with a concise description of these results. Specifically, we present these results in the context of their impact upon the program requirements and objectives, and their relevance to Phase II activity and ultimate commercialization. We now summarize these results by task.

In **Task 1**, we defined the SAVANT IF and satisfied all the task requirements. First, the IF essentially makes no semantic changes to the VHDL source, thus satisfying the first technical requirement, to preserve as much semantic content from the original source input as possible. Second, the IF was designed with particular attention to achieving extensibility. Hence, it is indeed extensible for the inclusion of additional CAD tool synthesized information, which satisfies our second technical requirement. Finally, by producing this IF definition, as described above and further elaborated in the Intermediate Form Description of Appendix A, we satisfied Project Performance Requirement 1, to establish the preliminary standard IF. As such, this IF definition served to partially achieve Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

The results from **Task 2** were aligned to no particular requirements, due to the re-scoping of this task as we described in Section 1.3. Hence, our efforts were essentially to begin defining how the SAVANT technology may be proliferated, and to stimulate initial community awareness of this technology. As a result, we formed the definition of how the SAVANT technology may be transferred to the design community, in the context of a licensing and distribution approach. We decided to distribute the SAVANT Analyzer (SCRAM) and IF definition freely (no charge) and easily available through the World Wide Web (WWW), and to provide a robust simulator based on SAVANT technology, also at no charge. Users of the SAVANT Analyzer/IF/Simulator will be allowed to create derivative products. In addition, we will allow distribution of derivative work for non-commercial purposes. However, for-profit distribution, support, rent, or lease of SAVANT-based technology will be allowed only upon completion of a profit-sharing agreement between MTL and the distributor.

By implementing this liberal licensing and distribution scheme, we will stimulate research in the VHDL community and continually extend its utility to the community. This should establish community acceptance by proliferation of the SAVANT technology and encourage development of its extensions. In

addition, we have begun the process of creating SAVANT awareness. MTL representatives have discussed possible utilization of SAVANT with several EDA vendors such as Exemplar, Synopsys, Intergraph, Mentor Graphics and Intermetrics at the Fall VIUF conference. EDA vendors were receptive to our ideas and have expressed an interest in obtaining a copy of the software and documentation. Through these results, we achieved Objective 2, to (begin to) establish community acceptance of SAVANT and to define the commercial product.

In **Task 3**, we analyzed available tools and selected the Purdue Compiler Compiler Tool Set (PCCTS) which satisfied our technical requirements to (1) analyze available tools for synthesizing compilers, and (2) select the appropriate tools for SAVANT. These accomplishments also satisfied Project Performance Requirement 5, to select the compiler support tools. We determined that PCCTS is an ideal and practical choice for SAVANT, as it:

A. Exceeds the requirements

B. Contains support for exception handling that facilitates error reporting and recovery

C. Integrates well with C++

D. Includes a grammar which successfully parsed over 1400 test files of VHDL models.

This selection of PCCTS also contributed, along with the results of Tasks 1, 4, and 5, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

In developing and testing the SCRAM analyzer, under **Task 4**, we essentially satisfied the *basic* technical requirements, to (1) Achieve a public domain analyzer, (2) Have archive capabilities, (3) Employ tools which are stable, reliable, and well documented, and (4) Make the software and documentation as portable as possible. As such, this also satisfied Performance Requirement 6, to demonstrate the capability of the compiler support tools by producing the prototype SCRAM. However, we note that a *complete* IF definition is not finished. The added design problems posed by the novel solution required additional design and implementation effort not foreseen when the original proposal and work definition was written. The novel solution (see section 2.4) required the construction of additional parts for exploring or demonstrating capacity of an extensible, object-oriented IF. As we discuss in Section 2.4, the benefits are that a better and more efficient final solution, that is more easily (and subsequently, more readily) integrated by the research CAD community, will result.

Through this development of SCRAM, with its inherent demonstration of the compiler support tools, we provided the following results:

A. Demonstrated the ability to parse VHDL '93.

B. Implemented many IF nodes.

C. Implemented some extensions to rewrite concurrent statements as process statements (which was actually not part of the original Phase I proposal).

D. Implemented several methods to regenerate VHDL from the IF.

This development of SCRAM contributed, along with the results of Tasks 1, 3, and 5, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

Under **Task 5**, we determined that VHDL would be an appropriate record and playback file format. As such, we satisfied the technical requirements for record and playback functions, implemented in a manner which would not interfere with the IF or standard exchange goals of the project. Here, we satisfied Project Performance Requirement 7, to establish the initial record and playback file formats. By defining this VHDL file format, we produced the following results:

A. Decided to use VHDL as intermediate file format

B. Established and integrated the library structure with the SAVANT GUI front-end

C. Although not yet integrated with analyzer, established that the basic functionality is already present: (1) by definition, the analyzer inputs VHDL, (2) the publish_vhdl() methods already generate VHDL for the intermediate format, and (3) the publish_vhdl() methods output to a redirectable file.

This decision to use a VHDL format for record and playback satisfied Project Performance Requirement 7, as well as the particular technical requirements for this task. As such, this selection contributed, along with the results of Tasks 1, 3, and 4, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

The result from **Task 6**, is the product documentation, consisting of this Final Technical Report, and the appended SAVANT IF Definition and Product Plan. The requirements were simply to produce this documentation, which we have done. The IF definition, along with the SCRAM analyzer, compiler support tools, and record and playback format definition, form the preliminary design basis for Phase II implementation. The Product Plan is our approach to commercialization, or technology transition, of SAVANT into the commercial

sector. In aggregate, this documentation is the substance of satisfying all of the program objectives, as we have described under the other task discussions of this section, and elaborate in Sections 2 and 3.

Under our (additional) **SAVANT Documentation Definition Task**, we defined the SAVANT system (or product) documentation framework and elements, and evaluated documentation support candidates. Here, we investigated the four critical aspects to effective SAVANT documentation, which are outline construction, tool gathering, IF documentation, and translation to other formats. This definition forms the basis of the documentation needs for the Phase II implementation as well as for the commercial product to be derived from the SAVANT technology. We now summarize the results under each of these aspects.

Outline Construction: Our requirement here was to construct a framework or outline for SAVANT documentation. The results from our investigations were outlines of the five main chapters, consisting of Terms and Conditions for copying, distribution and modification, Overview of SAVANT, Object-Oriented SAVANT, Integrating with CAD Tools, and The Intermediate Form which is a description of the in memory representation of each VHDL type. The user will need to know the intermediate form if SAVANT is to be used. These results serve to satisfy our requirements in several ways. First, we have identified the main components of the SAVANT documentation. Also, we have provided the user with the terms for using SAVANT, a definition of SAVANT, a summary of why SAVANT was developed and how to use SAVANT, and finally what information a user needs to know when using SAVANT.

Tool Gathering: The requirement for tool gathering included the analysis, retrieval, and testing of tools which would allow for the development of SAVANT documentation. The result was obtaining a typesetting tool called *TeX*, the documentation macro *Texinfo*, and the postscript figure macro *psfig.tex*. These results serve to satisfy our requirements in several ways. First, these tools produce two formatted outputs without the aid of additional tools. The first output is a *.dvi* file that can be used to generate a hardcopy. The second type of output is an *info* file. An info file is a hypertext file that can be viewed in Gnu's *Emacs* or by using a stand alone hypertext reader called *info*. We tested these tools by first incorporating the outline in Texinfo. We successfully printed a hardcopy and an info file from this outline.

IF Documentation: The requirement for IF documentation was that of displaying an easy-to-understand view of the intermediate form for VHDL as produced by SAVANT. The results were developing a hierarchy of classes currently included in SAVANT, collapsing each inheritance branch of the VHDL type classes and writing the data member information as nodes in Texinfo. These results serve to satisfy our requirements in several ways. The method of collapsing the inheritance branches of the hierarchy tree provides a systematic way of obtaining the intermediate form. By creating nodes in Texinfo for each VHDL type intermediate form, we obtained a simple way of displaying this information to the user. In the hypertext files, when a user is viewing a VHDL type intermediate form, the user can chose to view the classes associated with this intermediate form. In the hardcopy, these classes will be in the index.

Translation: The requirement of the translation aspect was translate the one source document into multiple output documents. The results from our investigation into the translation aspect were recognizing exactly what output files need to be generated from the source. We have found a substantial number of users of the World-Wide Web (WWW), so we made it a point to product a HTML (HyperText Markup Language) file from the Texinfo source document. These results serve to satisfy our requirements in several ways. First, we have discovered the two most prevailing media for distributing documentation — Postscript and HTML. By using a WWW browser (such as Mosaic) a user can view documentation in a hypertext environment. If, however, the user requires a hardcopy, this can be available via FTP (File Transfer Protocol) which is a common method for retrieving files from other computers who reside on the Internet. We also satisfy the one source document translation to multiple output documents requirement by showing successful translated files.

Our documentation definition results served to further the design definition of SAVANT as a whole, thus contributing to achieving Objective 3, to produce a preliminary design concept. These results also supported achieving Objective 2, to establish community acceptance, by defining the nature of the SAVANT documentation to be proliferated among the community.

In ***conclusion of this Results Summary of Section 1.4***, our results provided the required feasibility validation, Phase II foundation, and commercialization product plan necessary for a successful Phase I effort. This also concludes our introductory section (1.0). In the next section (2.0), we describe the specific technical activities which contributed to these results.

2.0 TECHNICAL INVESTIGATIONS

In this section, we provide an elaborated discussion of our technical investigations, with attention to the particular technical achievements and their relevance to meeting the project goals and objectives. We have organized our discussions in the context of the individual tasks. For each task in this section, we (1) describe the technical requirements we established, (2) outline the methodology we applied, (3) relate any significant events or discoveries resulting from the execution of the methodology, and (4) provide a summary of the investigations.

In our task activities we focused upon the key technical contribution of the SAVANT program — to establish a standard intermediate form of digital systems for the exchange of electronic data among CAD tools. In general, the intermediate form representation of a digital system design can be input from a variety of sources (textual languages, graphical languages, etc.); however, the primary source for this effort, presuming a Phase II implementation, will be the DoD standard hardware description language VHDL. Thus, in addition to designing and documenting the intermediate form, the Phase II SAVANT implementation and derivative commercial product will include a VHDL-to-intermediate form translator. The intermediate form will be an in-memory tree data structure. Consequently, SAVANT will also require some mechanism for off-line archiving and retrieving of digital system designs represented in the intermediate form. Finally, the SAVANT project must address the problem of technology insertion; how will the industrial, government, and academic communities be encouraged to use the SAVANT technology? These issues are more fully discussed below.

2.1 Task 1 — Establish the Preliminary Standard IF Definition

In Task 1, we were to apply our proposed concepts for the standard IF. The result of this task was to be first a "draft" standard IF definition to be shared with the design community and iterated into the deliverable Phase I-level standard IF definition, to satisfy Performance Requirement 1 (Establish preliminary standard IF).

Technical Requirements: The importance of and need for a standard intermediate form was discussed in Section 1.0. Briefly, a standard intermediate form is important and will serve the design automation community by providing a unifying, easy to process, representation of electronic designs. That is, instead of analyzing, verifying correctness, and manipulating VHDL source, CAD tools will be able to interface with an intermediate form of the design that has already been analyzed for syntactic and static semantics correctness.

The design of an intermediate form must preserve as much semantic content from the original source input as possible. However, it is not necessary to retain an ability to exactly reproduce the source input. That is for example, comments, newlines, and spaces are not language constructs with semantic content (while semantic constructs can be added as comments, cf VAL/VHDL, the VHDL language does not formally relate semantic content with comments). Thus, some information from the original source input may be discarded.

While the intermediate form will not preserve *all* information from the original source input, it should allow for the augmentation of the design data by CAD tools. More precisely, a CAD tool may need to mark components of the intermediate form with additional information for later use (by the same or other CAD tools). For example, a simulation code generator may need to decorate the intermediate form with code templates for later phases in the code generation process. Thus, the intermediate format must be extensible for the inclusion of additional CAD tool synthesized information.

Methodology: Our methodology was simple and straightforward, and designed to take maximum advantage of previous accomplishments. It consisted of the following steps:

A. Study existing solutions which may offer potential.

B. Review the status of past standardization efforts for IFs, as well as any procedural interfaces which may have been defined.

C. Design and implement any solutions, standardizations, or procedural interfaces deemed potentially useful in an analyzer's working IF nodes.

D. Review or examine the use of these IF methods in contemporary CAD tools.

E. Iterate steps C and D to focus upon useful and valid methods to be implemented in the SAVANT IF design.

Through this methodology we expected to be efficient in applying useful prior technology and effective in homing in on an effective IF definition. Next, we

describe some of our experiences in executing this methodology.

Task Execution: In the course of conducting this task, while identifying potentially viable techniques as planned, we noted a particular significance in the context of *object-oriented approaches.* In particular, we noted that past solutions have *claimed* to be object-oriented but lacked a defined ability for extensibility. We also realized that object-oriented design allows the possibility that nodes can be augmented with data and (overloaded) methods. Hence, self-definition of an object-oriented structure is great boon to CAD tools since the [overloaded] method is automatically resolved, which appears to eliminate the need for a procedural interface. Furthermore, since the IF can be made extensible, the augmenting data can be strongly typed. While all of these aspects will contribute to a highly-effective standard IF, they also require careful design of any software tools for implementing the standard IF. So the benefits are desirable, but demand care in their implementation.

Task 1 Summary: In summary, our venture into the object-oriented issues allowed us to conceive a novel solution, allowing a *fully extensible* IF definition. Although this was a valuable discovery, the entire problem studied in Phase I was enlarged somewhat by our novel solution, indirectly causing a less complete development of the IF in Phase I than we had anticipated. However, the result is a much cleaner, more flexible final design, which will ultimately produce a more effective implementation in a Phase II development, as we describe further in our description of task results in Section 3.1.

2.2 Task 2 — Interact with the Community

Under Task 2, we were to interact with the community, including users, vendors, and particularly the DASC. The result of this task would be the useful knowledge and community acceptance necessary to ensure a valid and acceptable standard IF. Additionally, this task would define community needs for a SAVANT Support Environment, which we viewed as the commercial product to be derived from this program. Task 2 would satisfy Performance Requirements 2, 3, and 4 (Establish community acceptance, vendor participation, DASC sub-committee).

In the course of the program, however, we, in concert with our WL/ELED sponsors, decided to de-emphasize this task. Our decision was based upon the opinion that it would be more appropriate to further the SAVANT research in Phase I before aggressively promoting or proliferating the technology within the

community. Hence, we took a more-or-less passive approach, consisting of only discussing SAVANT within the VIUF framework, and determining a methodology for distributing and proliferating the SAVANT technology, to be implemented in Phase II.

Due to this de-scoping of Task 2, we have no explicit requirements or methodology to discuss here, and no particular execution issues to describe. We basically "spread the word" in a fairly casual manner, responded frankly to any inquiries, and defined the proliferation methodology, the results of which we describe in Section 3.2.

2.3 Task 3 — Select Compiler Tools

Tasks 3 and 4 were to be a tightly-integrated pair. Together, their execution would satisfy Performance Requirements 5 and 6 (Select tools and construct prototype SCRAM). In Task 3, we would begin with our experience with the laboratory version of SCRAM at UC, and the draft standard IF definition from Task 1. Under this task, we would select the appropriate compiler tools to construct the deliverable prototype SCRAM under Phase I.

Technical Requirements: As previously mentioned, the construction of a VHDL analyzer is a complex problem. In fact, this problem is sufficiently complex that it prevents many research investigations from reaching a full integration with VHDL. Even the problem of merely forming a machine-processable set of grammar productions for VHDL is quite difficult. Despite much interest and many queries, little progress has been made toward the construction of a public domain VHDL parser. The chief problem is that the grammar given in the language reference manual is written primarily for human consumption and does not easily translate to a machine-processable form. In fact, most attempts at building a VHDL parser fail because most available compiler-compiler tool-sets produce parsers with only one token look-ahead and an LL(1) or LR(1) grammar for VHDL is difficult to construct.

In this task, our requirement was to analyze available tools for synthesizing compilers and to select tools for SAVANT. Because the intent of the SAVANT project is to publicly release all source code for CAD research and experimentation, the compiler support tools must also be publicly available and redistributable. Thus, only tools that are freely available in the public domain were considered.

Methodology: For the analyzer portion of this task, we located and copied all of the public domain parser generators announced in the monthly posting of comp.compilers. The parser generators were all examined for their suitability for a VHDL analyzer. In addition to reviewing the capabilities of each of the parser generators, we conducted an analysis of the available grammars for VHDL.

Task Execution: We executed this task by taking advantage of Internet access to a wealth of tools and grammars. More specifically, we first obtained, through FTP, comp.compilers monthly documents announcing public domain compiler development tools. This provided us a comprehensive source list for these tools. Next, again through FTP access, we obtained certain tools, selected from the overall list, for evaluation. These included yacc/lex (and bison/flex), pccts, eli, and coco. Similarly, we queried the comp.lang.vhdl community for available VHDL grammars, then evaluated available grammars and tools. Here, we concentrated upon VHDL grammars available for yacc/lex and pccts.

From these candidates, we then made our selection based upon the requirement for using effective, yet public-domain items. Here, the level of PCCTS development and presence of VHDL '93 grammar for PCCTS prompted its selection for SAVANT. We elaborate upon this result in Section 3.3.

Task 3 Summary: In summary of our approach to this compiler tool selection task, by obtaining a significant and valid candidate list from qualified sources, and evaluating our candidates in the context of our requirements, we were able to make the proper selections.

2.4 Task 4 — Demonstrate Tools and Build Prototype

As previously mentioned, Tasks 3 and 4 were to be a tightly-integrated pair. Together, their execution would satisfy Performance Requirements 5 and 6 (Select tools and construct prototype SCRAM). In Task 4, we would proceed to develop and test the prototype SCRAM, supplementing it with the final standard IF definition produced by Task 1, as the task proceeds. The results of Task 4 would be the deliverable prototype SCRAM and the test IF used to confirm its functionality, documented to a preliminary design-level of detail.

Technical Requirements: The SAVANT IF requires support tools before it can be widely accepted and inserted into the CAD research community. Consequently, support tools to build and manipulate the SAVANT IF must be available. At a minimum, a public domain analyzer and IF definition must be available. In

addition, archive capabilities must be available. These tools must be stable, reliable, and well documented for integration by CAD tool researchers and developers. The software and documentation must be as portable as possible — not requiring any special purpose hardware or expensive software packages for use. Initially, the software will be developed for UNIX based workstations.

Methodology: Our methodology for this task was precise and direct. It was simply to (1) build a grammar that processes *all* available VHDL models, (2) augment this with the software necessary to create the IF nodes, and (3) construct preliminary IF extensions for manipulation and output generation (hence evaluating the support for CAD tool construction). We encountered no significant problems in executing this methodology, as we describe next.

Task Execution: In executing this task, we accomplished the necessary technical activities to support the methodology. First, we refined the PCCTS grammar, building a preprocessor to distinguish quotes from character literals (a limitation of the lexer tool of PCCTS required this fix). Then, we repaired some bothersome, pathological VHDL parsing problems (e.g., name'('a')). Having achieved these grammar and parser "tune-ups," we then tested the implementation against available VHDL models and fixed bugs as they appeared. Here, approximately 1400 test VHDL models were successfully processed. Next, we built the C++ actions necessary to construct the IF nodes into the grammar. Finally, we built extensions to the IF to demonstrate the capabilities for CAD tool integration. Specifically, these extensions included rewriting VHDL concurrent statements to process statements and implementing a code generator to regenerate VHDL statements from the IF.

Task 4 Summary: Our Task 4 approach to demonstration and prototyping was successful. Here, the grammar was completed and we were able to process a significant number of VHDL models. Also, many IF nodes were implemented and tested. Rewriting and output generation was implemented to validate the benefits of object-oriented design which were made in our Task 1 activity (see Section 2.1).

2.5 Task 5 — Establish File Format for Record and Playback

In Task 5 we were to establish the record/playback format, to satisfy Performance Requirement 7. The result was to be a tested format capable of supporting the archiving needs of the IF, and documented to a preliminary-design level of detail.

Technical Requirements: The VHDL analyzer will translate VHDL into the intermediate form. The intermediate form is a memory resident data structure (tree) that must be archived into some library format for later use. That is, VHDL design units (design entities, packages, etc.) must be analyzable and storable into a design library for later use by other VHDL design units. Therefore SAVANT will also include two additional functions called RECORD and PLAYBACK to archive the intermediate form. RECORD will save the intermediate form representation of a VHDL design unit into the design library and PLAYBACK will read a design unit from design libraries into the intermediate form. For simplicity, initial implementations for RECORD/PLAYBACK will simply use VHDL as the library format. Later implementations may improve on this. However, SAVANT will not be severely limited by the capabilities of an implementation of RECORD and PLAYBACK. That is, the chief issue is maintenance of the integrity of the intermediate format. RECORD/PLAYBACK will be required to maintain the intermediate format; the format used for archival storage will not interfere with the intermediate format/standard exchange objectives of this project.

Methodology: Our Task 5 methodology included technical review, solution design, and consideration for future change, in the context of the record and playback formats. It consisted of three parts:

A. A review of the particular requirements for VHDL library management

B. The design of a solution that supports quick prototyping of the tools described in Task 4 (Section 2.4).

C. Maintaining the ability to replace an implementation with more efficient file formats, should they become available.

Task Execution: In executing this methodology, we encountered no particular problems. Hence, we proceeded to study existing library solutions of the vendor community, then select the appropriate file format and library index structure for our needs.

Task 5 Summary: Based upon our investigations, we decided to use a simple record and playback format structure that operates as an extension of the IF node structure, to serve our immediate needs. The file format was simply VHDL written by the publish_vhdl() modules. SCRAM (the analyzer) was reused to input the library structures. Although this solution is slower to process than a regular file format, it provided a quick solution which can be easily replaced as time permits in a subsequent Phase 2 implementation. The results from testing this implementation are given in Section 3.5.

2.6 Task 6 — Document the Phase I Effort

The documentation aspect of the Phase I effort evolved into a two-faceted endeavor. Under Task 6, we planned to assimilate all information regarding designs, discoveries, and community interaction into this document, a Phase I Final Report. This report was to also contain a product insertion plan for transfer of the SAVANT technology into the commercial sector. Additionally, while executing the other Phase I tasks, we became aware of the need to define some tools, formats, translation means, and an overall framework for the ultimate production and distribution of supporting documentation for the Phase II SAVANT implementation. We felt that since such documentation support definition would be a critical factor in a successful Phase II endeavor, we should expend some Phase I resources to achieve it. Hence, we added a task to define the documentation framework and elements, and proceeded to evaluate documentation support candidates. These added task activities are described in Section 2.7.

Technical Requirements: Our requirement for the final report was that it should be accurate, all-inclusive, and in the form and format prescribed for a Phase I Final Report. For the product insertion plan, our requirement was that it should be practical, realistic, and in a form which may easily grow into a product management plan in Phase II.

Methodology: Our methodology for the final report and appendices was simply to gather and assimilate the results of the various technical tasks as they became available, and begin the integration and editing process when such volume of material warranted these actions.

Task Execution: In executing this task, the assimilation of task information proceeded more or less according to plan.

Task 6 Summary: In summary, we accomplished the accumulation of technical material and the production of this Final Report, IF Definition (Appendix A) and accompanying Product Insertion Plan (Appendix B) as planned.

2.7 SAVANT Documentation Definition (Added Task)

As previously mentioned, we added a task to the Phase I effort, to define the SAVANT system (or product) documentation framework and elements, and to evaluate documentation support candidates. These added task activities are described in this section.

There are four critical aspects to effective SAVANT documentation, which are outline construction, tool gathering, IF documentation, and translation to other formats. The relationship among these aspects is illustrated in Figure 4. As shown, each input, with the exception of the outline construction aspect, is a generated output of the previous aspect. For each aspect, a requirement must be satisfied in order to complete the aspect given. These requirements are labeled as inputs in the figure. The outputs identify results of the aspects. After the translation aspect is accomplished, the end result is obtained in both hypertext documents and hardcopy documents.

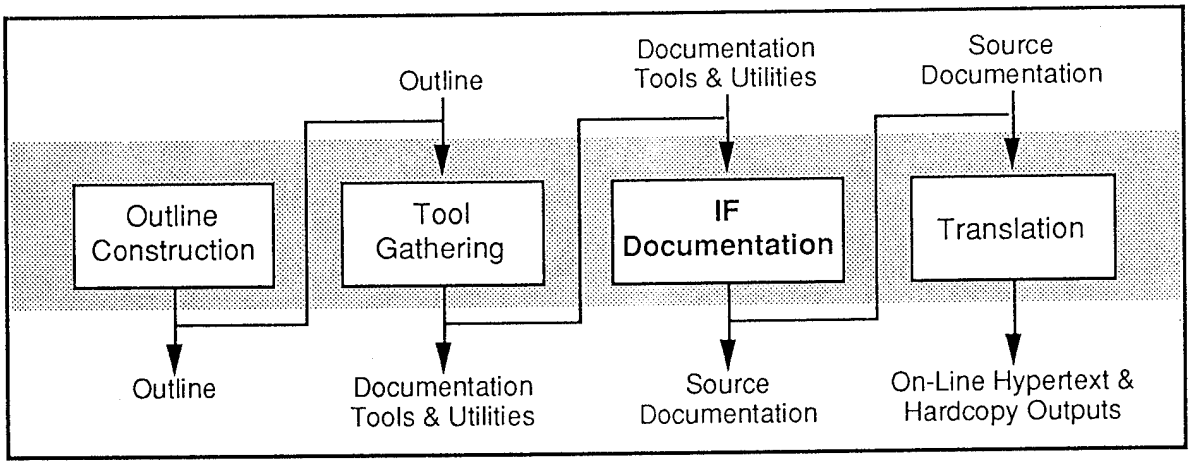

Figure 4. Aspect Relationships in SAVANT Documentation

These aspects are particularly critical to a successful SAVANT implementation because they identify a process by which the documentation for SAVANT will be generated. Each aspect performs a task on the given input, but the task details are changeable. For example, a change in the outline construction causes a change in the outline content, but not a change in format. Therefore, this change has no impact on the tools used to process the outline. To clarify, a change in content in any input to one of our aspects does not change the format of the input. In the ensuing discussions, we describe our Phase I technical investigations into these critical documentation aspects. For each aspect, we first discuss the requirements, then describe the methodology of our investigations and our execution of that methodology.

Outline Construction: We begin with the outline construction aspect, which is to construct an outline for the SAVANT documentation.

Requirements: The requirements for the outline construction were to identify the main components of the SAVANT documentation, decide what information is needed, and for whom this information is intended. Their basis was to formulate a complete outline of the SAVANT documentation for a particular set of end users. They also were to recognize the scope of information needed for completeness.

Methodology: We designed our methodology to ensure meeting the requirements. As such we investigated the typical end user of SAVANT. We analyzed the end uses of SAVANT in determining the type of information that will be used by the end users of the documentation.

Execution: In executing our methodology we generalized our end users to those with a high degree of VHDL and computer knowledge. We also assumed a minimum level of knowledge in Object Oriented programming. The end users range from university students to professional CAD tool developers. The outline therefore eliminates the need of having chapters relative to such topics as introducing VHDL or elementary C++. One problem with developing an outline such as this is the dynamic property of the assumed user. For example, if a user from a different field of study finds use in SAVANT, the documentation should be understood by this user. However, our assumptions limit the user set to a general group that may or may not include the new user.

Tool Gathering: The tool gathering aspect includes the analysis, retrieval, and testing of tools which will allow for development of SAVANT documentation.

Requirement: The requirement for tool gathering was to locate tools and utilities to provide a formatted output of the SAVANT documentation.

Methodology: We designed our methodology to ensure meeting the requirement. In doing so, we established a criteria for which tools are to be used. This criteria was developed as a result of outlining a few constraints. The first constraint was to use tools that are currently in the public domain. This constraint allowed public use of any portion of the documentation (outline, source documentation, and IF documentation) without the downloading of specific tools from us. Second, we recognized an importance in the localization of one source document. The benefit in having one source is that changes in documentation need only appear in one source. Third, we needed a formatting tool that has many utilities in the public domain to translate one

source to different, and unique, outputs. From the given constraints, we had to locate and retrieve a tool that adheres to the constraints as well as be easy to use. In testing the tools, we needed to determine if content material affects file format. If so, the tool or tools would be discarded.

Execution: In executing our methodology we found an abundant amount of tools in the public domain. It was difficult to decide the proper tools given many had translation utilities and were easy to use. In order to decide on the proper tools, we checked the archives of many locations and found which tools were in a wider circulation than others. This was a determining factor in the tools chosen.

IF documentation: The aspect of IF documentation includes displaying an easy to understand view of the intermediate form for VHDL as produced by SAVANT.

Requirement: The requirement for IF documentation was to provide a systematic way of documenting the intermediate form from SAVANT. It had to be simple enough to understand but thorough enough to encompass the complete intermediate form.

Methodology: In designing our methodology to satisfy the requirements, a translation had to occur between what SAVANT put into memory (as data structures) to documenting these structures. We first needed a hierarchy of the classes used in SAVANT in its current state. From this hierarchy, we could reduce the code to data only. This data, as it applies to VHDL syntax and semantics, could be extracted and written in its declaration form. What this implies is that a user who reads the documentation of the IF can easily recognize the data structures as easily as viewing a data structure in C or C++. This process is shown in Figure 5.

Execution: In executing our methodology we developed the class hierarchy with relative ease. It was surprising to find simplicity in developing a graphical hierarchical view of SAVANT's classes. From the hierarchy we collapsed the inheritance properties and extracted data members only. We documented these data members and arranged them according to their respective VHDL type. In order to minimize efforts in the remaining sections of the documentation, we also documented the class interfaces.

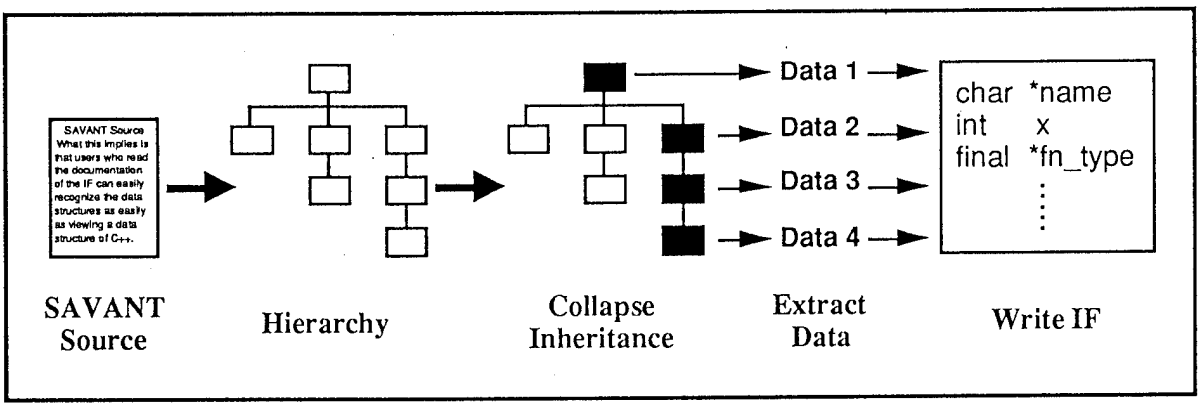

Figure 5. IF Documentation Method

Translation: The final aspect in documentation is translation, which deals with the media as presented to the user. The responsibility here is to translate the one source document into multiple output documents.

Requirements: The requirements for the translation were to decide what the final documentation should look like and what form it should have. First, every final document should maintain the integrity of the original source documentation. Second, every final documentation item should be produced by tools from the public domain. Finally, the final documentation should be accessible.

Methodology: We designed our methodology to ensure meeting the requirements. As such, we analyzed current media for information retrieval. From these media, we were to locate tools from the public domain to do translations from the output of the IF documentation into files that can be used by the media. If none existed, we were to locate alternative translation utilities. We were to verify that information in the source documentation was to match the information of the new files. To test the output from the translation aspect, we were to download these files in a manner similar to a user and validate the information.

Execution: In executing our methodology, we found that an exact duplication from source document to a final output file in differing media could be accomplished if the information was *text* only. Translating graphics provided a challenge in that different media requires different graphic formats. We discovered two main types of information retrieval. The first type is a hardcopy based file format (generally postscript). The second type is an on-line hypertext environment where certain words or objects have linking capabilities to other words, pages, or chapters. The hardcopy

involved a linear traversing of information, thus the writing needed to reflect this in the manner of chapters, indexes, and reference tags. In a hypertext environment, the writing needs to follow a manner more associated with nodes and the linking must have order and direction. In our selection of tool for developing the source documentation we have solved this problem. Our tool for developing the source documentation allows the writing to be formatted into nodes, which translate into chapters, sections, and even indices.

Documentation Definition Investigations Summary: Our investigations in this task proved to be successful and provided the necessary definition of the outline, tools, IF documentation, and translation means. The specific results of this task are discussed in detail in Section 3.7.

2.8 Summary of the Technical Investigations

In summary of our technical investigations, we have described the requirements, methodologies, and significant execution aspects of all tasks within our Phase I program. Our IF definition activities diverged from the original course into the realm of object-oriented aspects - and provided a cleaner design as a consequence. Our community interaction was useful and appropriate for this level of SAVANT development. For compiler tool selection, by obtaining and evaluating a significant and valid candidate list, we were able to make the proper choices. Our demonstration and prototyping was successful, as the grammar was completed and we were able to process a significant number of VHDL models. We decided upon a simple record and playback format structure that operates as an extension of the IF node structure, to serve our immediate needs. We accomplished the accumulation of technical material and the production of this Final Report and accompanying Product Insertion Plan as planned. Finally, we produced the necessary definition of the SAVANT documentation aspects — outline, tools, IF documentation, and translation means. Next, in Section 3.0, we discuss the results we obtained from these approaches, and how they served to satisfy our requirements.

3.0 PHASE I RESULTS

In this section, we provide an elaborated discussion of our Phase I results, defining their extent toward solving the problem and forming a foundation for subsequent Phase II activity. As in Section 2, we have organized our discussions in the context of the individual tasks. For each task in this section, we:

A. Describe the results we obtained from our technical investigations, experiments, or other analyses.

B. Describe how these results pertain to satisfying the technical requirements asserted in Section 2 as well as the Project Performance Requirements of Section 1.

C. Provide a summary of these results in the context of how they served to achieve our Program Objectives, which were also given in Section 1.

3.1 Task 1 — Establish the Preliminary Standard IF Definition

In Task 1, we were to execute a particular methodology to satisfy certain technical requirements as we described in Section 2.1, to begin development of the standard IF. The result of this task was to be first a "draft" standard IF definition to be shared with the design community and iterated into the deliverable Phase I-level standard IF definition, to satisfy Performance Requirement 1 (Establish preliminary standard IF). Our technical requirements were that:

A. The design of an intermediate form must preserve as much semantic content from the original source input as possible.

B. The intermediate format must be extensible for the inclusion of additional CAD tool synthesized information.

In the remainder of this section, we describe the results we obtained, how they served to satisfy the project and technical requirements, and how these accomplishments relate to achieving the objectives of the program.

Results Obtained: During Phase I, we explored several aspects of the objectives for this task. Most significantly, we reviewed different aspects of object-oriented representations and decided that an object-oriented design would be most suitable

for the IF. This decision also has been followed by several others in their design, however, our approach differs significantly in that our design allows for an extensible class definition within the object oriented representation. Thus, the CAD researcher can augment the class hierarchy with additional data and methods for problem-specific needs. Hence, decoration of the IF with additional data, information, or functionality is well-supported and, furthermore, the CAD researcher directly benefits from the fact that the IF is object-oriented (and, self-defining). That is, the CAD researcher benefits from all aspects of an object-oriented representation such as inheritance, polymorphism, and encapsulation. Also, we have also discovered an implementation technique for C++ that will fully support this design abstraction. Preliminary demonstration of this functionality has been successfully accomplished during Phase I, as we describe below in greater detail.

The SAVANT IF definition is designed as an object-oriented data structure with each node in the tree derived from a common base object. The intermediate form will be an extensible definition that is capable of adding additional data members and methods to each node in the intermediate form. This derivation structure is shown in Figure 6. This figure is shown for illustrative purposes only and should be considered a partial definition of the final derivation tree (In particular, the actual design has considerably more intermediate class definitions. This abbreviated tree is shown to help simplify the example for discussion purposes.)

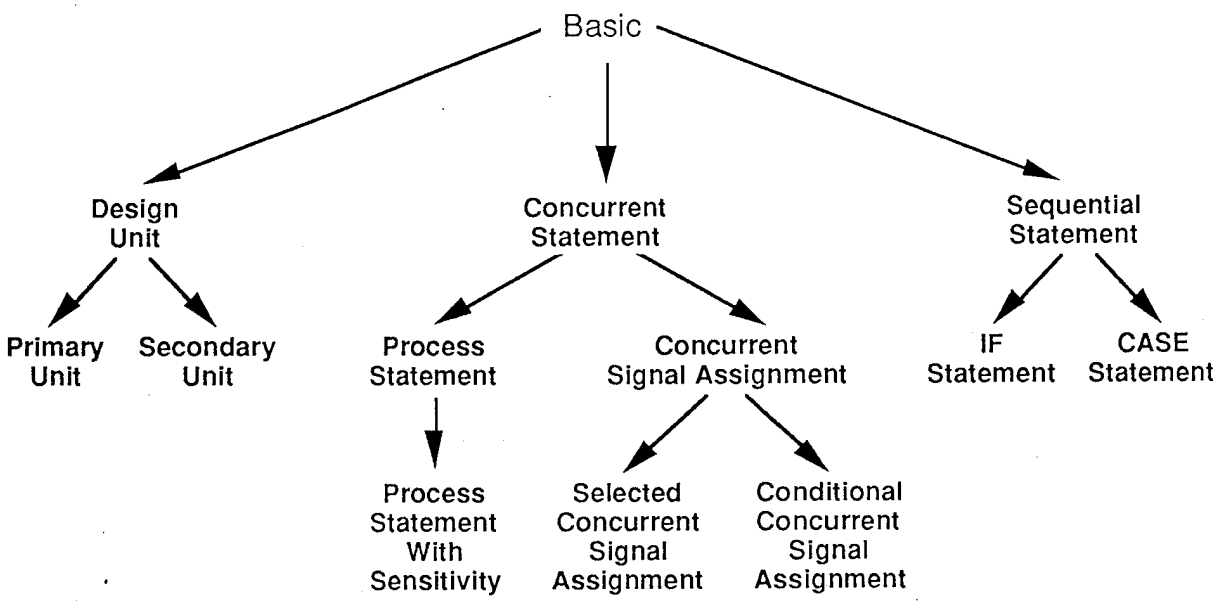

Figure 6. Derivation Structure of the IF

An example of how the basic intermediate form is extensible is shown in Figure 7. In this figure, the nodes inside the shaded area are the base intermediate form definition. The nodes outside the shaded area illustrate what might be used for a simple code generator (cgen).

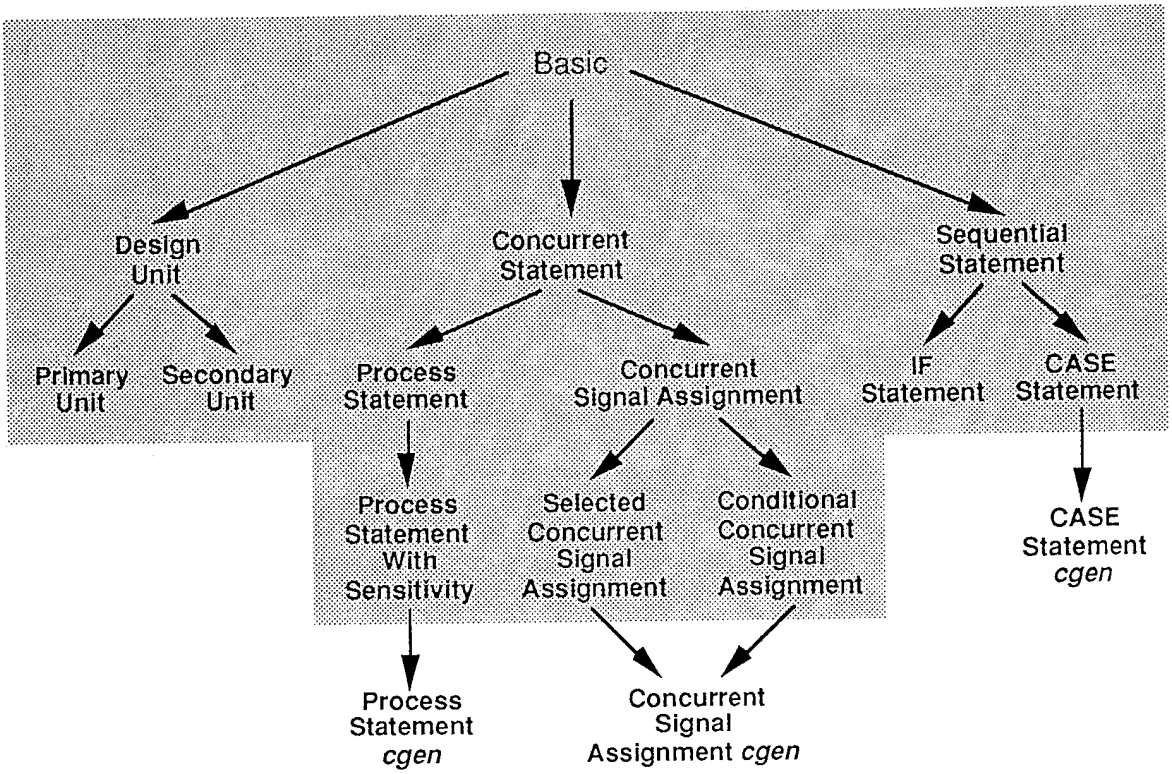

Figure 7. Extensibility of the IF

Figures 6 and 7 also show the logical organization of the desired intermediate form. The software implementation accompanying SAVANT that builds the intermediate form will be written in C++ and will require some additional structure to achieve the desirable functionality. In particular, the implementation will follow a structure as shown in Figure 8. In this figure, the basic intermediate form is captured by the nodes in the shaded area. Other nodes such as those needed for research CAD tools are shown outside of the shaded area. Four important observations need to be made about this figure.

1. The base node of the intermediate form class derivation tree is actually derived from base nodes for each of the research CAD tools.

2. In instantiating new nodes for the intermediate form, only those nodes shown in the shaded area at the bottom of the structure are to be created. This is enforced by having a single procedure called create-node defined in the base class that actually performs all node creation.

3. The leaf nodes of the intermediate form must be maintained as the research CAD classes and are added to the intermediate form. This is necessary so that constructors/destructors are invoked and methods/data of the intermediate classes become known.

4. Intermediate nodes in the intermediate form may also have classes derived by the research CAD tools. The reason for this is explained in Section 3.4 (see the discussion of the publisher/transmute classes to be included with the initial SAVANT software release).

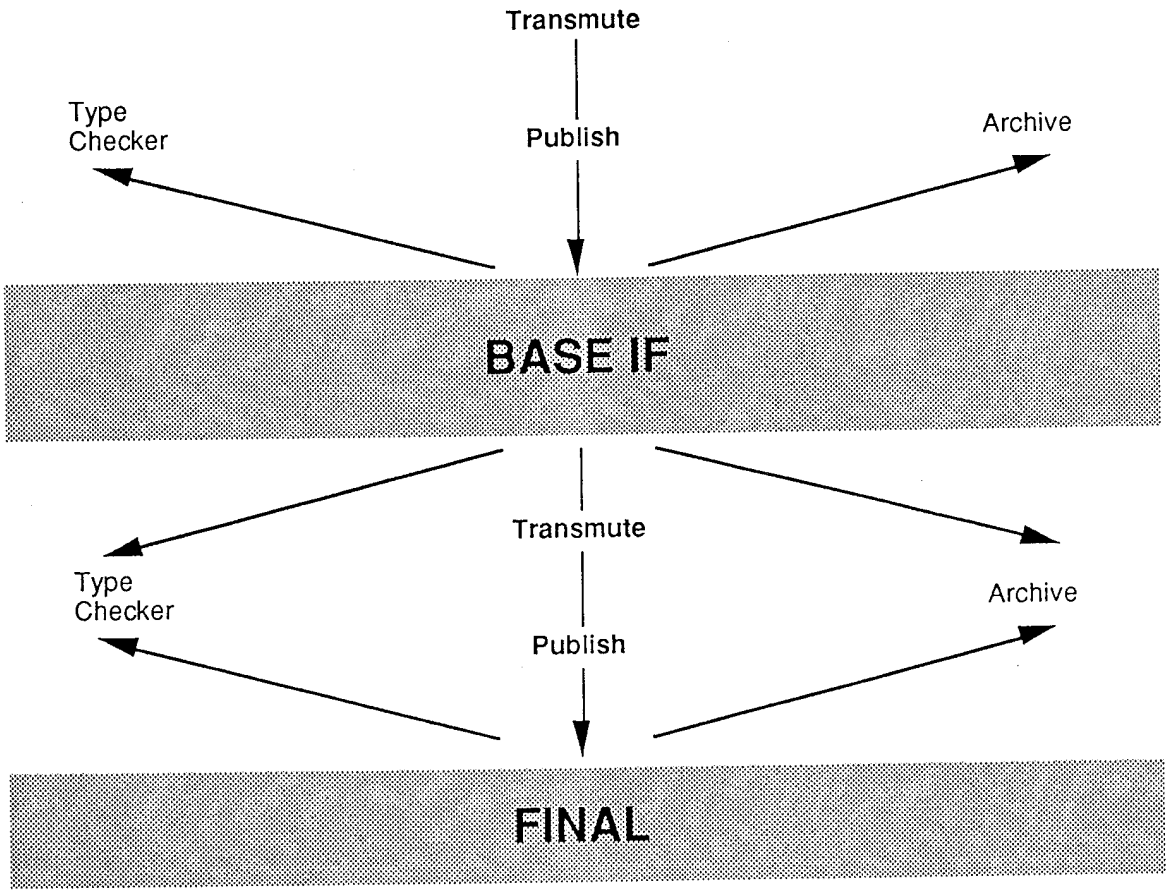

Figure 8. Software Implementation Structure

Satisfying the Requirements: In defining the IF as described above, we satisfied all the requirements for this task. First, the IF essentially makes no semantics changes to the VHDL source, thus satisfying the first technical requirement, to preserve as much semantic content from the original source input as possible. Second, the IF was designed with particular attention to achieving extensibility. Hence, it is indeed extensible for the inclusion of additional CAD tool synthesized

information, which satisfies our second technical requirement. Finally, by producing this IF definition, as described above and further elaborated in the Intermediate Form Description of Appendix A, we have satisfied Project Performance Requirement 1, to establish the preliminary standard IF.

Task 1 Results Summary: The IF definition we achieved satisfied Project Performance Requirement 1, as well as the particular technical requirements for this task. As such, this IF definition served to partially achieve Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts). The other aspects necessary to completely achieve these two objectives are the selection and demonstration of compiler support tools and the definition of a record and playback file format. We describe our accomplishments in these areas later in Section 3.

3.2 Task 2 — Interact with the Community

As we discussed in Sections 1.4 and 2.2, we, in concert with our USAF sponsor, decided to de-scope this task to (1) fairly passive discussions of the SAVANT technology within the VIUF, and (2) defining the means for its proliferation and distribution. In this section, we concentrate upon the results of these two activities.

Results Obtained: The results from this task form the definition of how the SAVANT technology may be transferred to the design community, in the context of our determined licensing and distribution approach and activities to stimulate awareness. We have decided to distribute the SAVANT Analyzer (SCRAM) and IF definition freely (no charge) and easily available through the World Wide Web (WWW) to anyone and everyone who wants it. In addition, we will also provide a robust simulator based on SAVANT technology at no charge through WWW. Users of SAVANT Analyzer/IF/Simulator will be allowed to create derivative products. In addition, we will allow distribution of derivative work for non-commercial purposes. However, for-profit distribution, support, rent, or lease of SAVANT-based technology will be allowed only upon completion of a profit-sharing agreement between MTL and the distributor.

By implementing such a liberal licensing and distribution scheme, we will stimulate research in the VHDL community and continually extend its utility to the community. Our strategy, stated simply, is to establish community acceptance by proliferation of the SAVANT technology and encouraging development of its extensions.

MTL will profit by providing products that enhance the utility of basic SAVANT technology. Such products would include an interactive and fast simulation environment, a man-machine interface for SAVANT and derivative products developed by third party vendors. In addition, MTL would be able to profit from a pay-per-use service based on SAVANT.

In addition, we have begun the process of creating SAVANT awareness. MTL representatives have discussed possible utilization of SAVANT with several EDA vendors such as Exemplar, Synopsys, Intergraph, Mentor Graphics and Intermetrics at the Fall VIUF conference. EDA vendors were receptive to our ideas and have expressed an interest in obtaining a copy of the software and documentation.

Satisfying the Requirements: In general, this activity had no explicit technical requirements.

Task 2 Results Summary: As a result of this task, we defined the technology distribution technique and created some initial SAVANT awareness within the community. These actions will ultimately assist us in the implementation of our Product Plan, which is described in Section 3.6 and elaborated in Appendix B. They also served to satisfy Objective 2, to (begin) to establish community acceptance and to define the commercial product.

3.3 Task 3 — Select Compiler Tools

In Task 3, we were to execute a particular methodology to satisfy certain technical requirements as we described in Section 2.3, to select the appropriate compiler tools to construct the deliverable prototype SCRAM under Phase I. The result of this task was to be these selected tools, to satisfy Performance Requirement 2 (select compiler support tools). Our specific technical requirements were to:

A. Analyze available tools for synthesizing compilers.

B. Select the appropriate tools for SAVANT.

In the remainder of this section, we describe the results we obtained, how they served to satisfy the project and technical requirements, and how these accomplishments relate to achieving the objectives of the program.

Results Obtained: The findings from our technical investigations, as we discussed in Section 2.3, were that the Purdue Compiler Construction Tool Set (PCCTS) provided an excellent support environment for the SAVANT analyzer development effort. PCCTS supports an extended BNF (EBNF) notation and inputs LL(k) grammars. Inherited and synthesized attributes, parser exception handling, token classes, and lexical classes are all supported by PCCTS. The software is in the public domain and runs on a variety of platforms including SUN, DEC, SGI, VAX, HP, Linux, NetBSD, MSDOS, and OS/2. Furthermore, the VHDL grammar input to PCCTS was originally developed at UC and is the only available grammar in the public domain that supports VHDL '93.

Thus, we plan to use the PCCTS compiler construction toolkit to build the VHDL analyzer. PCCTS generates LL(k) parsers and was selected over other tools such as YACC/LEX or Cocktail because (i) it is LL(k), (ii) it support predictive parsing, (iii) it readily supports attribute transmission, and (iv) it builds a parser compatible with C++. Because VHDL designs can easily grow quite large and because LL parsers tend to be slightly faster and more compact, we believe that PCCTS is an excellent choice for SAVANT's VHDL analyzer. Furthermore, the PCCTS developers are actively engaged in extending the tool-suite and are planning to incorporate extensive error recovery capabilities in 1995. Finally, PCCTS generates ANSI C that is processable by g++. Consequently, we propose to build the parser actions in C++.

Satisfying the Requirements: In analyzing available tools and selecting PCCTS, as we described above, we satisfied our technical requirements to (1) analyze available tools for synthesizing compilers, and (2) select the appropriate tools for SAVANT. These accomplishments also satisfied Project Performance Requirement 5, to select the compiler support tools.

Task 3 Results Summary: In summary, PCCTS is an ideal and practical choice for SAVANT, as it:

A. Exceeds the requirements

B. Contains support for exception handling that facilitates error reporting and recovery

C. Integrates well with C++

D. Includes a grammar which successfully parsed over 1400 test files of VHDL models.

This selection of PCCTS satisfied Project Performance Requirement 5, as well as the particular technical requirements for this task. As such, this

selection contributed, along with the results of Tasks 1, 4, and 5, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

3.4 Task 4 — Demonstrate Tools and Build Prototype

In Task 4, we were to execute a particular methodology to satisfy certain technical requirements as we described in Section 2.4, to develop and test the prototype SCRAM, supplementing it with the final standard IF definition produced by Task 1, and demonstrating the compiler support tools as a natural consequence of the development. The result of this task was to be the deliverable prototype SCRAM and the test IF used to confirm its functionality, documented to a preliminary design-level of detail, to satisfy Performance Requirement 6 (demonstrate capability of the compiler support tools by producing the prototype SCRAM). Our specific technical requirements for the SCRAM analyzer and tools were that:

A. A public domain analyzer must be available.

B. Archive capabilities must be available.

C. The tools must be stable, reliable, and well documented for integration by CAD tool researchers and developers.

D. The software and documentation must be as portable as possible — not requiring any special purpose hardware or expensive software packages for use.

In the remainder of this section, we describe the results we obtained, how they served to satisfy the project and technical requirements, and how these accomplishments relate to achieving the objectives of the program.

Results Obtained: In this task, we constructed the basic elements of a VHDL '93 analyzer and many of the initial class definitions for the IF. In particular, we have a full VHDL '93 grammar that inputs to PCCTS. The resulting parser correctly parses over 1400 test files. However it has no semantic testing. In addition, we constructed many classes for the object-oriented representation. The classes are organized into three components, namely: base nodes, CAD tool nodes, and leaf nodes. The base nodes contain all the data and method definitions for the standard IF definition. The leaf nodes are dummy classes that are used by the parser to create IF objects. The leaf nodes are derived from the CAD nodes and allow for user-added constructor/destructor invocations as well as for a search up the derivation tree for the correct implementation of virtual functions.

The CAD tool nodes are organized into two parts. The first part contains pure virtual functions and serve as base classes from which the common base node for all of the standard IF nodes are derived (thus making the virtual functions visible). The second part is the actual implementations of the functions for each node in the standard IF. Thus, for example, a code generator CAD tool would define a pure virtual function cgen() from which the base standard IF node would be derived. The remaining standard IF nodes would have a derived class containing an implementation for cgen(). Lastly, the leaf classes would be modified to derive from the classes for cgen() and the desired extensibility is accomplished.

This functionality was completely demonstrated (but not fully implemented) in the Phase I effort. In addition, we showed the implementation of two functionalities. First, we implemented a transmute() method that rewrites the IF nodes for concurrent statements into an IF node (and descendants) for the equivalent process statement. Second, we implemented a publish_vhdl() method that outputs VHDL. Furthermore, we added a publish_vhdl() method derived from the concurrent statement IF node that automatically causes an invocation of the transmute() function. Thus, publish_vhdl() need not be defined for the nodes derived from concurrent statement and an automatic translation to a process statement IF node will be invoked. The publish_vhdl() method can then operate only on a subset of VHDL but actually achieve the desired capability across the entirety of VHDL. Figure 9 illustrates this concept.

The prototype SAVANT software system includes the following components integrated into the class derivation structure, as illustrated in Figure 8:

Scram: For translating VHDL source programs into the intermediate form.

Transmute: For manipulating the intermediate form and rewriting nodes from one form to another. In particular, the rewriting of concurrent statements into their equivalent process statement definition.

Publisher: Output routines that generate VHDL (publish-vhdl) and C++ (publish-cpp) representations of the intermediate form.

Archive: Library manager functions that load and store the intermediate form. Initially these functions will rely on VHDL as the intermediate form and invoke scram and publish-vhdl to read/write the library files.

In addition, a simple, public domain GUI for the system was constructed using TCL/TK. Named SAVANT, the GUI supports the control and invocation of the above described tools. It also supports the definition of library names and search paths to support the archive manager.

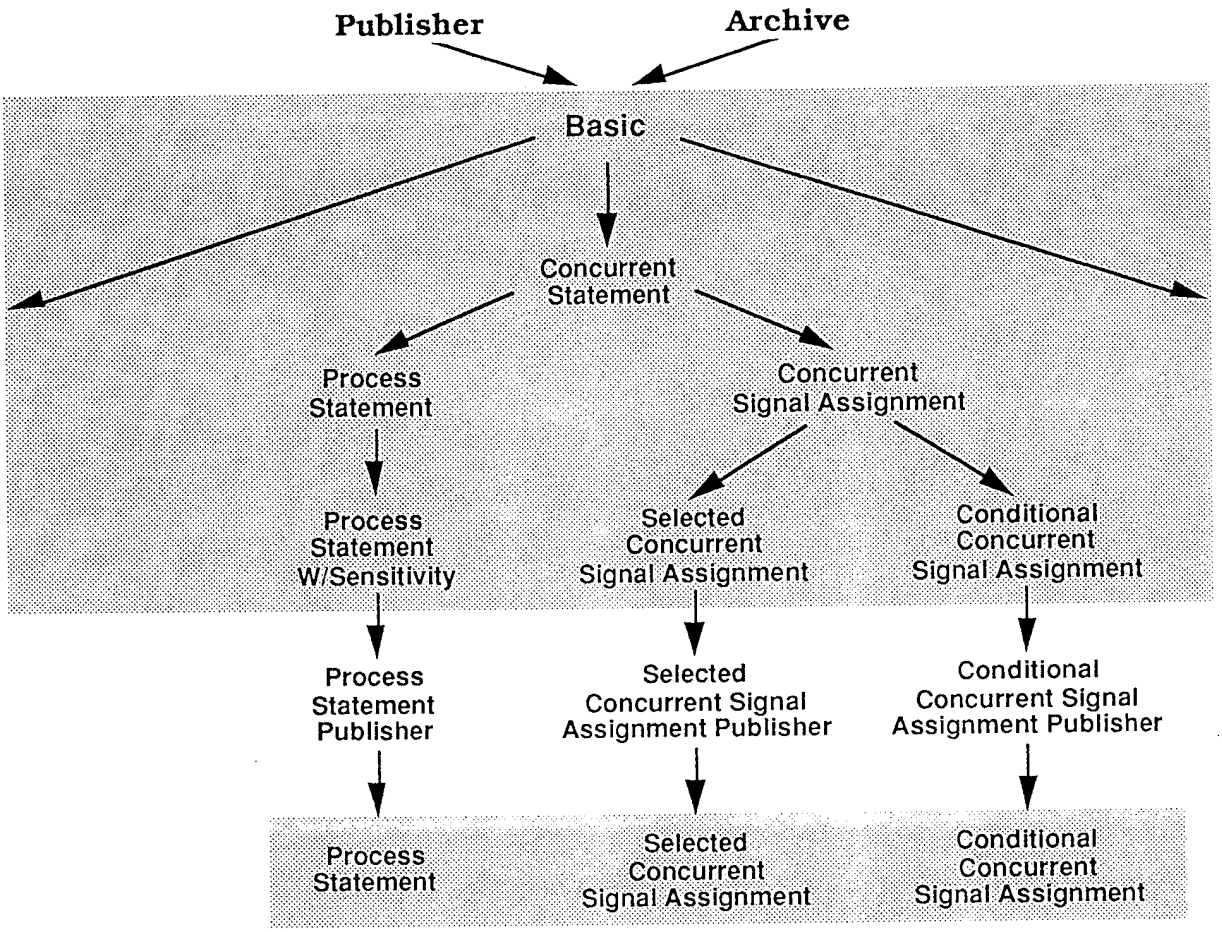

Figure 9. Prototype SAVANT Publisher

Satisfying the Requirements: In developing and testing the SCRAM analyzer, as described above, we essentially satisfied the *basic* technical requirements, to (1) Achieve a public domain analyzer, (2) Have archive capabilities, (3) Employ tools which are stable, reliable, and well documented, and (4) Make the software and documentation as portable as possible. As such, this also satisfied Performance Requirement 6, to demonstrate the capability of the compiler support tools by producing the prototype SCRAM.

However, we must note that a *complete* IF definition is not finished. The added design problems posed by the novel solution required additional design and implementation effort not foreseen when the original proposal and work definition was written. The novel solution (see section 2.4) required the construction of additional parts for exploring or demonstrating capacity of an extensible, object-oriented IF. As we discussed in Section 2.4, the benefits are that a better and more efficient final solution, that is more easily (and subsequently, more readily) integrated by the research CAD community, will result.

Task 4 Results Summary: In summary, through this development of SCRAM, with its inherent demonstration of the compiler support tools, we provided the following results:

A. Demonstrated the ability to parse VHDL '93.

B. Implemented many IF nodes.

C. Implemented some extensions to rewrite concurrent statements as process statements (which was actually not part of the original Phase I proposal).

D. Implemented several methods to regenerate VHDL from the IF.

This development of SCRAM satisfied Project Performance Requirement 6, as well as the particular technical requirements for this task. As such, this selection contributed, along with the results of Tasks 1, 3, and 5, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

3.5 Task 5 — Establish File Format for Record and Playback

In Task 5, we were to execute a particular methodology to satisfy certain technical requirements as we described in Section 2.5, to establish the record and playback functions. The result of this task was to be a tested format capable of supporting the archiving needs of the IF, and documented to a preliminary-design level of detail, to satisfy Performance Requirement 7 (establish initial record and playback file formats). Our particular technical requirements were that:

A. SAVANT will include two additional functions called RECORD and PLAYBACK to archive the intermediate form.

B. RECORD will save the intermediate form representation of a VHDL design unit into the design library.

C. PLAYBACK will read a design unit from design libraries into the intermediate form.

D. The format used for archival storage will not interfere with the intermediate format or standard exchange goals of this project.

In the remainder of this section, we describe the results we obtained, how they served to satisfy the project and technical requirements, and how these accomplishments relate to achieving the objectives of the program.

Results Obtained: Through the technical investigations described in Section 2.5, we have determined that initially the file format for SAVANT libraries will simply be VHDL. This allows us to exploit the publish_vhdl() method for both debugging and archiving. SCRAM itself will be used to input the library object. In addition, a simply library index is present to identify the library logical name and its contents. A library interface has also been incorporated into the SAVANT front-end GUI.

Satisfying the Requirements: By employing VHDL as the file format, as described above, we satisfied the technical requirements for record and playback functions, implemented in a manner which would not interfere with the IF or standard exchange goals of the project. As such, we also satisfied Project Performance Requirement 7, to establish the initial record and playback file formats.

Task 5 Results Summary: In summary, by defining the VHDL file format, as described above, we produced the following results:

A. Decided to use VHDL as intermediate file format

B. Established and integrated the library structure with the SAVANT GUI front-end

C. Although not yet integrated with analyzer, established that the basic functionality is already present: (1) by definition, the analyzer inputs vhdl, (2) the publish_vhdl() methods already generate vhdl for the intermediate format, and (3) the publish_vhdl() methods output to a re-directable file.

This decision to use a VHDL format for record and playback satisfied Project Performance Requirement 7, as well as the particular technical requirements for this task. As such, this selection contributed, along with the results of Tasks 1, 3, and 4, to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

3.6 Task 6 — Document the Phase I Effort

In Task 6, our requirement was simply to produce this final report, with the appended SAVANT IF Definition and Product Plan. We had no explicit requirements or methodology for this task. As a result, this report, the IF definition of Appendix A, and the Product Plan of Appendix B were produced.

3.7 SAVANT Documentation Definition (Added Task)

In this task, we were to execute a particular methodology to satisfy certain technical requirements as we described in Section 2.7, to define the SAVANT documentation for Phase II. The result of this task was to be appropriate definition of the documentation outline, tools, IF documentation, and translation means.

In the remainder of this section, for each particular aspect listed above, we begin with a concise statement of the requirements, and then describe the results we obtained and how they served to satisfy these requirements. We conclude with a summary of how these accomplishments relate to achieving the objectives of the program.

Outline Construction: We begin with the outline construction aspect, which is to construct an outline for SAVANT documentation.

Results: The results from our investigations into the outline construction aspect were outlines of the main chapters. There are five main chapters in the documentation for SAVANT. The first chapter is the *Terms and Conditions for copying, distribution and modification.* This chapter explains the legal issues surrounding the use of SAVANT. The second chapter is the *Overview of SAVANT.* It provides an introduction to SAVANT as well as identifying what SAVANT can be used for. The third chapter, *Object-Oriented SAVANT,* explains why we chose the object oriented paradigm for developing SAVANT. It shows the class hierarchy of SAVANT. The fourth chapter is titled *Integrating with CAD Tools.* This chapter explains to the CAD developer how SAVANT can be extended to existing CAD tools as well as methods for customizing SAVANT. The last chapter is titled *The Intermediate Form.* It is a description of the in-memory representation of each VHDL type. The user will need to know the intermediate form if SAVANT is to be used.

Satisfying the Requirements: These results serve to satisfy our requirements in several ways. First, we have identified the main components of the SAVANT documentation. We have provided the user with the terms for using SAVANT, what is SAVANT, why was SAVANT developed, how does a user use SAVANT, and finally what information a user needs to know when using SAVANT.

Tool Gathering: The tool gathering aspect includes the analysis, retrieval, and testing of tools which will allow for the development of SAVANT documentation.

Results: Our results from the tool gathering aspect were obtaining a typesetting tool called TeX, the documentation macro Texinfo, and the postscript figure macro psfig.tex. TeX is free and found in a wide number of archive sites, and can handle multiple languages in the same document. This is desired for the documentation of SAVANT as it provides a simple source document that can be translated into other documents for different media. Texinfo is a macro that can be included in the TeX file. Texinfo allows the development of nodes. Nodes are blocks of information that can be translated into chapters, sections, or not printed at all in the hardcopy. Nodes can also be linked by hypertext links in hypertext documents. A node can contain menu items for inclusion into a hypertext file. A menu item can also be processed as a index, table of contents, or not at all in the hardcopy. The postscript figure macro psfig.tex is a macro also included into the TeX file that allows printing of a postscript image in hardcopy.

Satisfying the Requirements: These results serve to satisfy our requirements in several ways. First, the tools we have chosen produce two formatted outputs without the aid of additional tools. The first output is a *.dvi* file that can be used to generate a hardcopy. The second type of output is an *info* file. An info file is a hypertext file that can be viewed in Gnu's Emacs or by using a stand alone hypertext reader called info. We tested these tools by first incorporating the outline in Texinfo. We successfully printed a hardcopy and an info file from this outline.

IF Documentation: The aspect of IF documentation includes displaying an easy to understand view of the intermediate form for VHDL as produced by SAVANT.

Results: The results from the IF documentation aspect were developing a hierarchy of classes currently included in SAVANT, collapsing each inheritance branch of the VHDL type classes and writing the data member information as nodes in Texinfo. We also documented the class hierarchy in Texinfo which will be used as an index in the hardcopy, and links from the IF nodes in the hypertext documents. Thus, we have the intermediate form documented and cross references to the classes that hold that intermediate form available to the user.

Satisfying the Requirements: These results serve to satisfy our requirements in several ways. The method of collapsing the inheritance branches of the hierarchy tree provides a systematic way of obtaining the intermediate form. By creating nodes in Texinfo for each VHDL type intermediate form, we obtain a simple way of displaying this information to the user. In the hypertext files, when a user is viewing a VHDL type intermediate form, the

user can chose to view the classes associated with this intermediate form. In the hardcopy, these classes will be in the index.

Translation: We now present the last aspect of translation, which deals with the media as presented to the user. The responsibility here is to translate the source document into multiple output documents.

Results: The results from our investigation into the translation aspect were recognizing exactly what output files need to be generated from the source. We have found a substantial number of users using the World-Wide Web (WWW), so we made it a point to product a HTML (HyperText Markup Language) file from the Texinfo source document. To do this, we use a translation utility from info to HTML (info2html). This produces a document that has limited graphics capability, therefore we have added commands to the utility in order to provide the extended graphics capability. We successfully translated the outline, IF documentation, and hierarchy into HTML. We also have translated the outline into a postscript file using a utility (dvips) which prints the dvi file on a postscript printer.

Satisfying the Requirements: These results serve to satisfy our requirements in several ways. First, we have discovered the two most prevailing media for distributing documentation — Postscript and HTML. By using a WWW browser (such as Mosaic) a user can view documentation in a hypertext environment. If, however, the user requires a hardcopy in the form of a postscript file, this can be available via FTP (File Transfer Protocol) which is a common method for retrieving files from other computers who reside on the Internet. We also satisfy the one source document translation to multiple output documents requirement by showing successful translated files.

In ***conclusion of the documentation definition task results***, they served to further the design definition of SAVANT as a whole, thus contributing to achieving Objective 3, to produce a preliminary design concept. These results also supported achieving Objective 2, to establish community acceptance, by defining the nature of the SAVANT documentation to be proliferated among the community.

3.8 Summary of the Results

In summary of our results, they served to completely satisfy our project performance requirements and Phase I objectives, as we have indicated in the individual task discussions. Furthermore, the IF definition, along with the

SCRAM analyzer development, compiler support tools definition and demonstration, and record and playback format definition, form a solid and well-quantified and validated basis for a successful Phase II development. Finally, the product plan of Appendix B, along with the documentation definition and proliferation methodology promises a successful transition of the SAVANT technology into a viable commercial product.

This concludes our Section 3 discussions of the results from our technical investigations. In the next section, we present our Phase I conclusions and recommendations, which are, of course, based upon these results.

4.0 PHASE I CONCLUSIONS AND RECOMMENDATIONS

In conclusion of our Phase I project, we considered this to be a highly successful effort. Although our focus was altered somewhat from our proposed direction, the results were significant and served to satisfy the program objectives.

In defining the IF in Task 1, we satisfied Project Performance Requirement 1, to establish the preliminary standard IF, and partially achieved Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

Through our community interactions of Task 2, although somewhat de-scoped from the level we originally intended, we defined the technology distribution technique and created some initial SAVANT awareness within the community. These actions will ultimately assist us in the implementation of our Product Plan, and served to satisfy Objective 2, to (begin to) establish community acceptance and to define the commercial product.

By analyzing available tools, and selecting PCCTS, in Task 3, we satisfied Performance Requirement 5, to select the compiler support tools. This selection contributed to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

In Task 4 by developing and testing the SCRAM analyzer and demonstrating the compiler support tools, we satisfied Performance Requirement 6, to demonstrate the capability of the compiler support tools by producing the prototype SCRAM. This contributed to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

By establishing VHDL as the file format in Task 5, we satisfied Project Performance Requirement 7, to establish the initial record and playback file formats. This selection again contributed to achieving Objective 1 (establish technical feasibility) and Objective 3 (produce preliminary design concepts).

In Task 6, our requirement was simply to produce this final report, with the appended SAVANT IF Definition and Product Plan. As a result, this report, the IF definition of Appendix A, and the Product Plan of Appendix B were produced.

As a result of our additional task to define the SAVANT documentation, we

furthered the design definition of SAVANT as a whole, thus contributing to achieving Objective 3, to produce a preliminary design concept. These results also supported achieving Objective 2, to establish community acceptance, by defining the nature of the SAVANT documentation to be proliferated among the community.

In aggregate, we satisfied all our project requirements, and achieved our three program objectives, to (1) establish technical feasibility, (2) begin to establish community acceptance, and (3) produce preliminary design concepts. Based on this success in accomplishing these objectives, and in particular the level of definition of SAVANT achieved, we recommend that a Phase II development of the SAVANT Technology should be considered. SAVANT addresses an acknowledged and critical problem for VHDL-in-CAD, is demonstrably feasible, and has received sufficient investigation and preliminary development to judge a Phase II prototype development to be a fairly low-risk endeavor.

MTL Systems, Inc. and the University of Cincinnati have enjoyed meeting the technical challenges and producing the results of this Phase I effort. We look forward to further developing and commercializing the SAVANT technology in a Phase II program.

Appendix A

SAVANT Intermediate Format Documentation

1. The Intermediate Form

This chapter is a work in progress. At present, it currently documents the intermediate form for the SAVANT analyzer as it currently exists at MTL. It is updated as new versions of SAVANT are obtained.

1.1 Design Units

1.1.1 Design Unit List

The IF representation of the design unit list contains the following data definition:

```
list_of_design_units        design_unit_list
```

1.1.2 Design Unit

The IF representation of a design unit has the following data definitions:

```
char                        *name
final_context_clause_list   *context_clauses
final_library_unit          *library_unit
```

1.1.3 Primary Units

1.1.3.1 Entity Declaration

```
char                        *name
final_port_list             *port_list
final_generic_list          *generic_list
final_declaration_list      *declaration_list
final_conc_stmt_list        *stmt_list
```

1.1.3.2 Configuration Declaration

For a configuration declaration, the IF data structure has the following declarations:

```
char                        *name
char                        *parent
final_declaration_list      *declarations
final_block_configuration   *block_configuration
```

1.1.3.3 Package Declaration

For a package declaration, the IF data structure has the following declarations:

```
char                        *name
```

1.1.4 Secondary Units

1.1.4.1 Architecture Body

For an architecture body, the IF data structure has the following declarations:

char	*name
char	*parent
final_declaration_list	*declarations

1.1.4.2 PackageBody

For a package body, the IF data structure has the following declarations:

In development

1.2 Sequential Statements

1.2.1 Wait Statement

For a wait statement, the IF data structure has the following declarations:

char	*name
char	*label
final_signal_list	*signal_list
expression	condition
expression	timeout

1.2.2 If Statement

For an if statement, the IF data structure has the following declarations:

char	*name
char	*label
expression	*condition
final_seq_stmt_list	*then_stmts
final_seq_stmt_list	*else_stmts

1.2.3 Case Statement

char	*name
char	*label
expression	*selector
when_clause_list	when_clauses

1.2.4 While Statement

For a while statement, the IF data structure has the following declarations:

char	*name
char	*label
final_seq_stmt_list	*stmt_list
expression	*condition

1.2.5 For Statement

For a for statement, the IF data structure has the following declarations:

char	*name
char	*label
final_seq_stmt_list	*stmt_list
identifier	*iterator
discrete_range	*range

1.2.6 Exit Statement

For an exit statement, the IF data structure has the following declarations:

label	*destination

1.2.7 Next Statement

For a next statement, the IF data structure has the following declarations:

char	*name
char	*label
label	*destination
expression	*condition

1.2.8 Assertion Statement

For an assertion statement, the IF data structure has the following declarations:

char	*name
char	*label
boolean_expression	*condition
expression	*report
expression	*severity

1.2.9 Report Statement

For a report statement, the IF data structure has the following declarations:

char	*name
char	*label
expression	*report
expression	*severity

1.2.10 Signal Assignment Statement

In development

1.2.11 Variable Assignment Statement

For a variable assignment statement, the IF data structure has the following declarations:

In development

1.2.12 Procedure Call Statement

For a procedure call statement, the IF data structure has the following declarations:

In development

1.2.13 Null Statement

For a null statement, the IF data structure has the following declarations:

In development

1.3 Concurrent Statements

1.3.1 Block Statement

In development

1.3.2 Process Statement

For a process statement, the IF data structure has the following declarations:

```
char                    *name
char                    *label
final_declaration_list  *declaration_list
final_seq_stmt_list     *stmt_list
int                     postponement
```

1.3.3 Process Statement with Sensitivity List

For a sensitive process statement, the IF data structure has the following declarations:

```
char                    *name
char                    *label
final_declaration_list  *declaration_list
final_seq_stmt_list     *stmt_list
int                     postponement
final_signal_list       *snsty_list
```

1.3.4 Concurrent Signal Assignment Statement

For a concurrent signal assignment statement,the IF data structure has the following declarations:

In development

1.3.5 Concurrent Call Statement

For a concurrent call statement, the IF data structure has the following declarations:

In development

1.3.6 Concurrent Assertion Statement

For a concurrent assertion statement, the IF data structure has the following declarations:

```
char                *name
char                *label
boolean_expression  *condition
expression          *report
expression          *severity
```

1.3.7 Component Instantiation Statement

For a component instantiation statement, the IF data structure has the following declarations:

In development

1.3.8 Generate Statement

For a generate statement, the IF data structure has the following declarations:

In development

Appendix B

SAVANT Product Plan

SAVANT Product Plan

In this plan, we describe potential markets and how we anticipate that our Phase II SAVANT results and deliverables will form a basis for Phase III commercialization, and discuss our specific plans for such commercialization. We consider *commercialization* to be the crux of this issue, whether it is achieved through a formal Phase III effort or by other means, and we present it here in this context.

We have divided this plan into 3 parts. To begin, in Part 1, we describe the commercial and military markets we expect to reach with the SAVANT Technology. Next, in Part 2 we describe our anticipated Phase II results, and how these will form the basis for the (Phase III) research and development necessary to realize a viable commercial product from the SAVANT technology. Finally, in Part 3, we describe our specific plans for Phase III and other commercialization of the SAVANT technology.

Part 1. Commercial and Military Markets for the SAVANT Technology

The potential markets and applications for the SAVANT technology are both vast and significant, in the context of commercial as well as government endeavors. In this section we provide an overview of this potential, and then describe particular government and commercial uses.

The applications for SAVANT are vast simply because the use and proliferation of VHDL as a foundation for myriad design automation tools is growing exponentially. Design automation tools for software design, hardware design, concurrent engineering, hardware/software co-design, performance modeling, and rapid prototyping, to name but a few, represent areas where VHDL is the core beneath the specific tool. Significantly, within this tool developer community, there is a growing and prevailing notion that standardization is the key to success. In the recent past, individual tool vendors would choose a particular VHDL environment, such as Cadence, Synopsis, or Model Technologies upon which to build their products. Now, these vendors are beginning to see the value of common VHDL foundations and looking to creative and innovative support environments and their specific tool attributes as their road to success. This motivation is of course partially influenced by user dissatisfaction with having to purchase several, separate VHDL environments for their different tools. In other words, the user and vendor communities are both ready and anxious for the standardization which SAVANT will provide. The applications for SAVANT technology are also significant because, in addition to satisfying the standardization needs described above, the SAVANT standard will be freely available to limited-resource-bound researchers as well as to well-funded

vendors. This will integrate researchers tightly with commercial vendors, thus making future technology transition significantly more effective.

Hence, we see the SAVANT technology being welcomed by vendors, their customers, and the research community at large. This offers a thriving marketplace for the SAVANT technology, to support both commercial and government needs. In both sectors, SAVANT will be used to provide the foundation for the tools discussed above. These tools will, in turn provide the automated design support for large-scale digital (and analog, with VHDL-A forthcoming) system designs. Some examples of these systems in the commercial and government sectors include:

Commercial: Industrial control and process control systems, dedicated computing or virtual computing systems, communications network and network control systems, traffic control systems, automotive electronics, medical and industrial imaging systems.

Government: Avionics systems, fire control and battle management systems, reconnaissance systems, specialized computing or virtual computing systems, target recognition/cueing systems, digital signal processing systems, communications network and network control systems, air traffic control systems.

In summary, we expect the SAVANT technology will provide the standardization and software support needed to enable more widespread use of VHDL, standardization and commonality among tools, and to enable more VHDL-in-CAD research. This indicates vast and significant applications, and a large commercial and government marketplace for this technology.

Part 2. Phase II Results - The Foundation for Phase III R&D

We anticipate the initial SAVANT Phase II results will consist of certain software components as well as supporting documentation. The software release will include the following components and capabilities:

Scram: Translating VHDL source programs into the intermediate form.

Transmute: Manipulating the intermediate form and rewriting nodes from one form to another. In particular, the rewriting of concurrent statements into their equivalent process statement definitions.

Publisher: Output routines that generate VHDL publish_vhdl and C++ publish_cpp representations of the intermediate form.

Archive: Library manager functions that load and store the intermediate form. Initially these functions will rely on VHDL as the intermediate form and invoke scram and publish_vhdl to read/write the library files.

Debugger: Basic debugging utilities provided through a command line interface. The debugger will be derived from the object-oriented QUEST simulation kernel and code generator.

Interactive user I/F: Easy-to-use, man-machine interface for interactive simulation and animation. The simulator will be based upon the QUEST simulation kernel, QUEST code generator and SAVANT debugger. (See Section 4 for details about the QUEST project).

The first four software objects will be placed in the public domain and will be developed by our subcontractor, the University of Cincinnati. The last two software objects (for debugging and interactive simulation) will be commercial software, developed by and the property of MTL Systems, Inc.

Figure B1 illustrates how these Phase II results provide a foundation for subsequent Phase III R&D. As shown, these results may be broadly categorized as *software and document results*. Within the software category, we have the *public domain software*, which consists of Scram, Transmute, Publisher and Archive. Also within this category, we have the *marketable software products*, which include the Debugger and Interactive User I/F. The *documentation* category includes the User's Manual, Installation Guide, Reference Manual, IF Definition, Final Report, and the WWW I/F which enables distribution of this information, as well as the public-domain software, over the WWW. The purpose of a Phase III effort would be to bring these products to a more effectively marketable form, as we describe below.

The *public domain software* is considered to be both a basis for marketable services and a marketing tool. Hence, in a Phase III effort (or other commercialization activity), we would build a service support structure around these products, which would permit us to effectively and profitably sell support to those users who require more assistance than can be obtained through the software and documentation distribution. The software itself would become a

marketing tool in the sense that its distribution and proliferation would stimulate a need for the services mentioned above. Therefore, a Phase III effort would build upon the Phase II-level versions of these products, and provide such improvements or enhancements necessary to render them a more acceptable asset to the user community. The distribution accomplished in Phase II will no doubt stimulate some feedback from the community, and this would be the basis for such improvements or enhancements in this, as well as the other product areas.

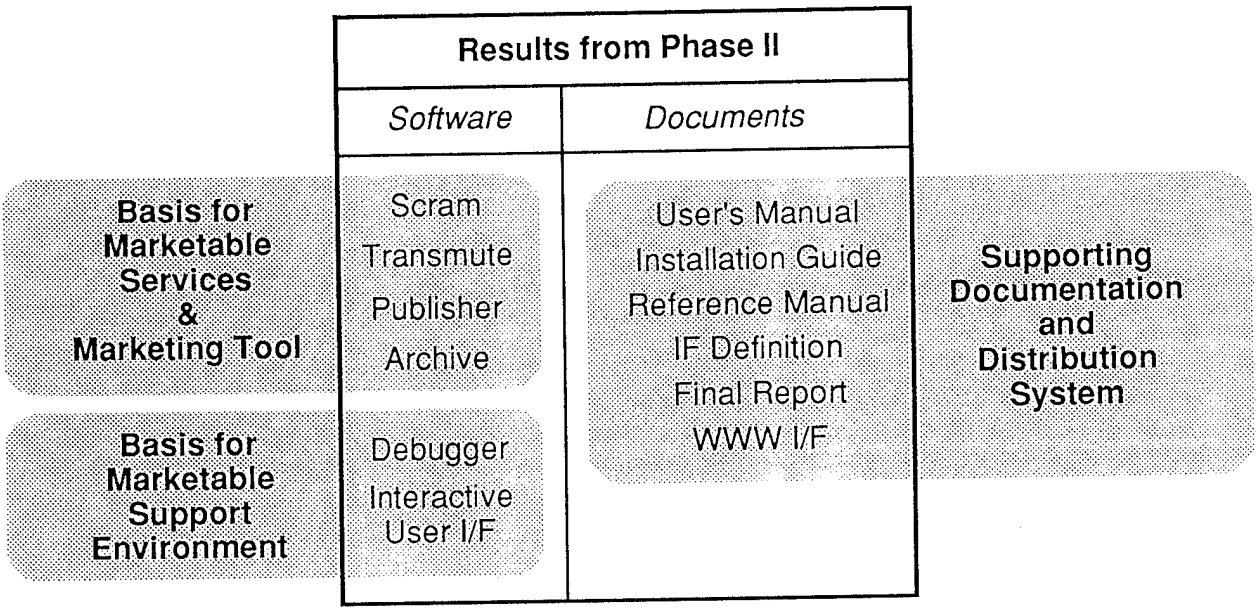

Figure B1. Phase II Results as a Phase III Foundation

The *marketable software products* from Phase II will form the basis for a user-friendly support environment, which would become the ultimate product here. In a Phase III or commercialization effort we would build upon this foundation, and add other support environment utilities, such as translators, file managers, additional visualization support, analysis tools, etc., to realize the complete SAVANT support environment product. Here, we would also put the software product support structure in place to provide for customer support and revisions throughout the products' life.

The *documentation* from Phase II would require periodic revisions throughout its lifetime. In a Phase III or commercialization effort, we would make the document modifications necessary to reflect any Phase III modifications to the software products. Here, we would also put the documentation support structure in place to permit additional revisions to be

effectively handled throughout the products' life, and to maintain the WWW distribution paradigm established in Phase II.

In summary, a Phase III or commercialization effort would build upon the Phase II products and comments from the user community regarding the Phase II release to create an integrated SAVANT Product Line, with the necessary support functions to ensure an effective and profitable product life. This concept is illustrated in Figure B2.

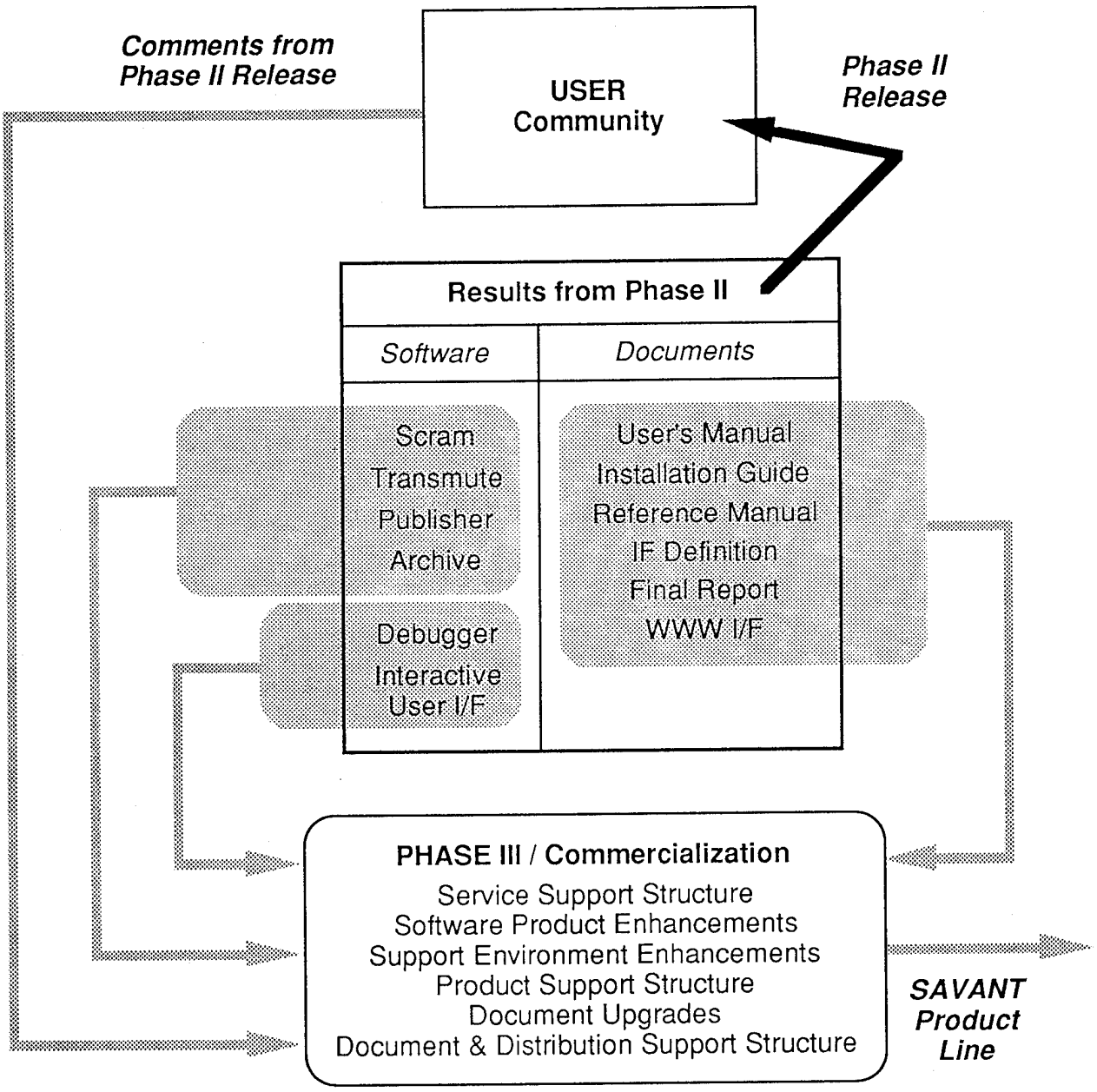

Figure B2. Creating the SAVANT Product Line

Part 3. The Phase III or Commercialization Plan

In this section we describe our specific plans for Phase III and other commercialization of the SAVANT technology. As we mentioned before, the Phase III work would be but a part of a longer-term and more extensive commercialization plan. This plan is designed to integrate several key ingredients: The Phase II Results, the Phase III developments discussed above, and the resources of MTL's INTELL**X** Center (an entity within the MTL corporate structure specifically tasked with commercializing technologies, especially for CAD and EDA). Our *general* commercialization approach is to synergize proliferation of the public-domain software with both a products and services business for long term marketing of the SAVANT technology. The elements of this approach are as follows:

DISTRIBUTION:

1. The SAVANT IF definition could be utilized for any purpose by anyone and would be freely distributed. A fee would be charged for the physical act of transferring the copy, which would always include the original copyright notice.
2. The SAVANT public-domain software and its derivative software could also be freely distributed by anyone else provided (a) all the source code is provided for SAVANT or its derivative, (b) the appropriate notice is included, (c) MTL Systems, Inc. is notified, (d) it is provided to everyone and anyone and (f) no fee is charged except for the physical act of transferring the copy.
3. Any distribution for profit including sale, support, lease and rent of SAVANT software or its derivative would only be permitted through negotiating a licensing or other profit-sharing agreement with MTL Systems, Inc.
4. WWW would be used by MTL Systems, Inc. to provide easy access to the SAVANT IF definition, public-domain software, and supporting documents.

COPYRIGHTS

The University of Cincinnati and MTL Systems, Inc. would hold the copyright for the SAVANT IF and software.

COMMERCIAL RIGHTS

1. MTL Systems, Inc. would hold exclusive commercial rights to the SAVANT software.
2. MTL Systems, Inc. would share its profits with the University of Cincinnati (to be negotiated between MTL and UC).

SIMULATOR

At least a parallel/uniprocessor simulator would be provided with restrictions similar to the SAVANT software and documentation, and would accompany the SAVANT analyzer. This simulator would be derived from the QUEST simulator.

SUPPORT SERVICES

1. Yearly support for the SAVANT software could be obtained for a fee from MTL Systems, Inc.
2. The support would be purchased at two levels: batch or interactive. Batch support would be obtained via email and voice mail. Interactive support would be obtained through phone or video conference.
3. The purchase of support would qualify the customer to any bug fixes and upgrades.
4. No other organization except MTL Systems, Inc. would be permitted to provide support for profit, unless a profit-sharing agreement is negotiated with MTL Systems, Inc.

INITIAL SAVANT-BASED PRODUCTS

1. An easy-to-use GUI could be purchased from MTL Systems, Inc.
2. An interactive User I/F could be purchased from MTL Systems, Inc.

PAY-PER-USE

1. SAVANT could be accessed on a pay-per-use basis from MTL Systems, Inc.

This distribution of public-domain entities, plus the marketing of products, for-profit services, and pay-per-use will enable the proliferation of the technology and the initial insertion of the SAVANT Product line into the marketplace, beginning in Phase II and continuing in a Phase III or other commercialization activity.

Figure B3 illustrates our overall, expected paradigm for commercializing the SAVANT technology, which includes the Phase II, Phase III/commercialization activities, and also considers the long-term product life. As we described in Part 2, the Phase II effort will result in initial versions of the products and an initial release or distribution of the public-domain entities. Then the Phase III or commercialization activities will produce the initial SAVANT product line, which will include the beginnings of the support services business. The configuration of this release will depend a great deal upon the level of investment we may be able to

obtain for commercialization actions here, as well as comments received from the user community on the Phase II-level release.

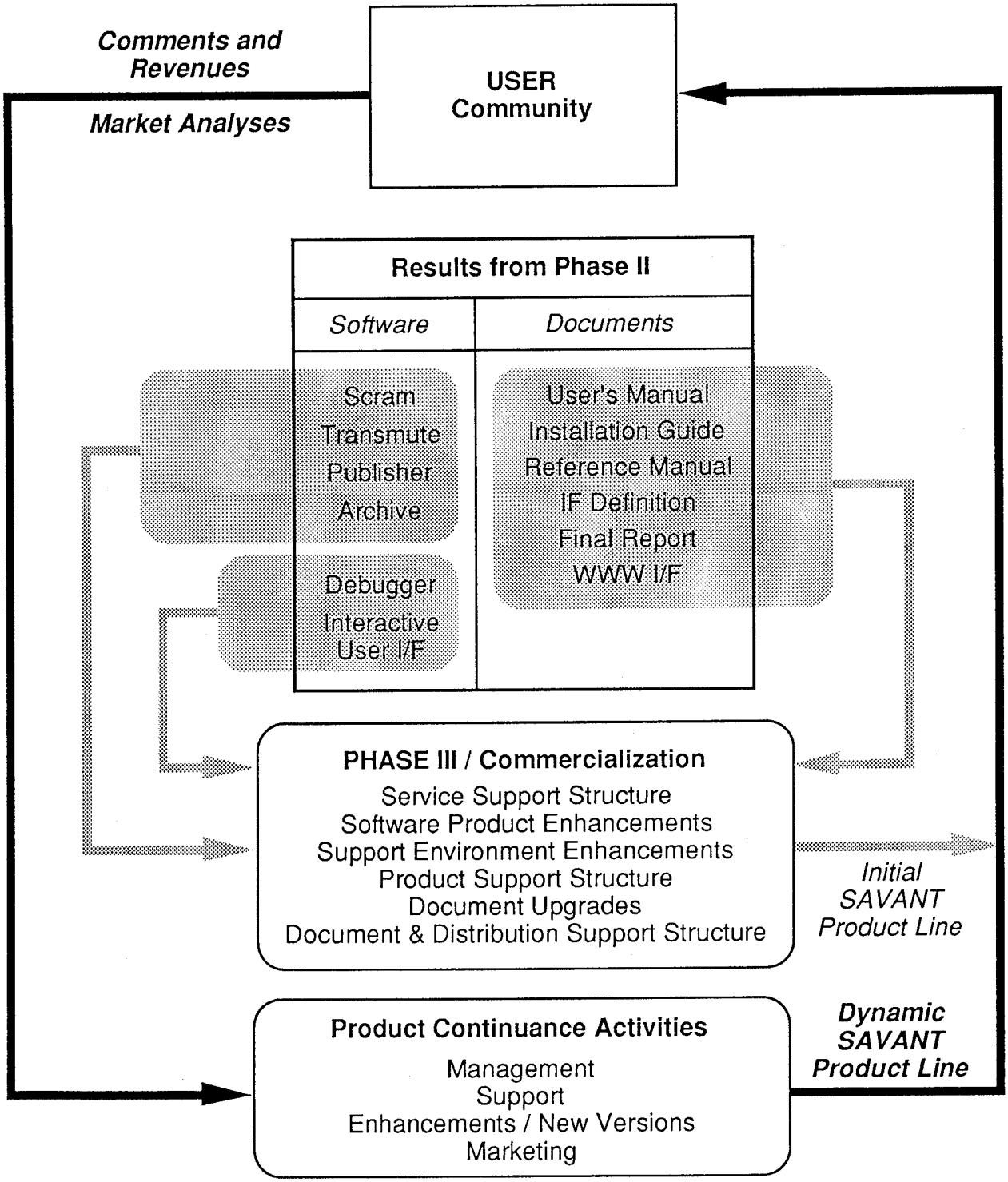

Figure B3. SAVANT Long-Term Commercial Product Paradigm

Long term operation of the SAVANT product line will be continual, dynamic, interactive cycles of product release, user community response, and product continuance activities. With each release, we will receive both comments and revenues from the user community. The revenues will fund the execution of the continuance functions of management, support, enhancements, and marketing. The support and enhancement functions will evolve from the Phase II/III implementations, while the management and marketing elements will be executed in a more organized and product-oriented manner than will have been done for Phase II/III activities.

The product-oriented attention and management of the SAVANT Product Line will be executed by MTL's INTELLX Center, a for-profit design automation tools & services center. INTELLX maintains high caliber talent and state-of-the-art tools to accelerate the productization of technology. We engage in EDA tool R&D, as well as electronic system design, analysis, and testing. Our engineers have direct experience with system performance analysis, hardware/software partitioning, and hardware/software project management from specification through assembly and test. We have worked on a variety of projects spanning industries such as ground vehicles, aerospace, computers, semiconductors, telecommuni-cations, consumer electronics, and medical equipment. Our engineers have designed and produced PCB's, hybrids, and semi-custom ICs. ProDESIGN is customer-driven, and committed to product & process excellence.

In ***summary***, our plan is to build effectively upon the Phase II effort to conduct a Phase III (or other) commercialization effort which will result in an initial SAVANT product line release. This activity and release will then integrate smoothly into a long-term product management plan under the direction of an organization whose sole purpose is to commercialize and maintain products for MTL. Through this approach we will ensure effective insertion of the SAVANT technology into the marketplace, whose demands and needs were summarized in the beginning of this plan.